

# 68

## MICRO JOURNAL

Australia A \$4.75 New Zealand NZ \$ 6.50  
Singapore S \$ 9.45 Hong Kong H \$23.50  
Malaysia M \$ 9.45 Sweden 30-SEK

**\$2.95<sub>USA</sub>**

### OS-9 Atari Amiga Mac S-50

6800 6809 68008 68000 68010 68020 68030

The Magazine for Motorola CPU Devices For Over a Decade!

This Issue:

Macintosh-Watch p.42

"C" User Notes p.7

Basically OS-9 p.17

FORTH p.45

Software User Notes p.21

OS-9 SK-DOS Atari Amiga  
FLEX Macintosh A User Contributor Journal And Lots More!

VOLUME X ISSUE IV • Devoted to the 68XXX User • April 1988

The Grandfather of "DeskTop Publishing™"

000422 A/E  
MR. MICKEY FERGUSON  
P.O. BOX 87  
KINGSTON SPRINGS TN 37082

SERVING THE 68XXX USER WORLDWIDE

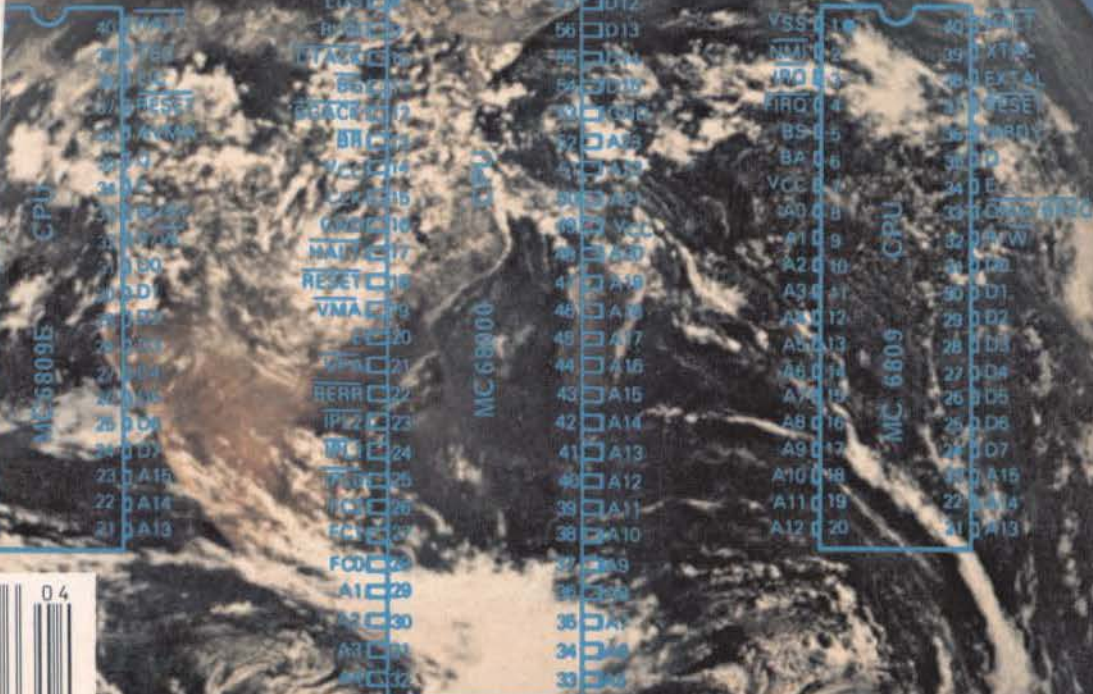


PHOTO CREDIT: NASA

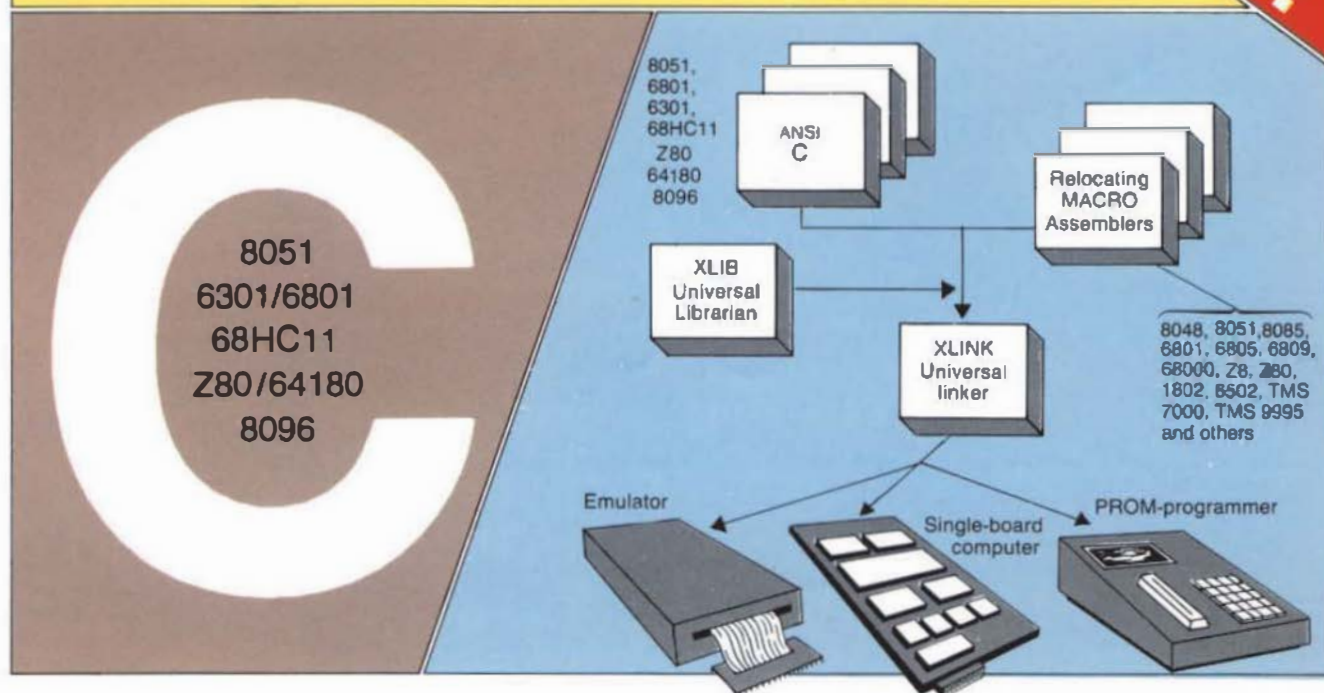


04

# ICC C CROSS COMPILER

## FOR OS-9/68xxx BASED SYSTEMS

**NEW!**



### ANSI C

Full implementation of the proposed ANSI standard for C compilers. Includes the Kernighan & Ritchie standard plus improvements for micro-controller development.

### Memory-based compiler

ICC is a fast one-pass compiler based on main memory storage. This has three major advantages:

- NO temporary files
- NO time-consuming assembly pass
- NO separate pre-processor stage

This combines into one single word: **SPEED**

### PROM-able C

ICC makes it possible to put C programs into PROM, still using the full C language and all data types, including type definitions, long integers and statically initialized variables.

### Built-in Type-Checking

ICC has a UNIX LINT-like type-checking option built-in into the compiler. This means that Pascal-like warnings are generated, e.g. when functions are mismatched or undeclared.

For more information contact your local distributor:

Frank Hogg Laboratory  
The Regency Tower  
Suite 215  
770 James Street  
Syracuse, NY 13203  
Phone: (315)474-7856  
Telex: 646740

Elsoft AG  
Zelweg 12  
CH-5405 Baden-Dättwil  
Switzerland  
Phone: (056) 83 33 77  
Telex: 828275

### Various Options

- 8051 — single-chip  
— 64 K CODE + DATA  
— 64 K CODE + 64 K DATA
- 6301 and 6801
- Z80 and 64180

### Full Package Development System

The ICC compiler package includes:

- C run-time library
- $\mu$ -Series Relocatable Macro Assembler
- XLINK Universal Linker
- XLIB Universal Librarian
- Floating-point support
- 150 page manual in three-ring binder

All this together give the micro-controller programmer a powerful Development System Software Environment.

**IAR**  
**SYSTEMS**

OS-9/68xxx version distribution by:

Micromaster Scandinavia AB  
Box 1309, S-751 43 UPPSALA, SWEDEN.  
Tel. int.: + 46 18 13 85 95. Telex: 76129



# GMX MICRO-20 and TWINGLE-20 PRICE LIST

**All versions include 1 SAB Board**

	MICRO-20 with 1MB RAM	MICRO-20 with 2MB RAM	TWINGLE-20 with 4MB RAM
12.5 MHz	1855.00	2155.00	3855.00
16.67 MHz	2185.00	2485.00	4185.00
20 MHz	2585.00	2885.00	4785.00

## OPTIONAL PARTS AND ACCESSORIES

68881 12.5MHz Floating Point Coprocessor	\$ 165.00
68881 16.67MHz Floating Point Coprocessor	\$ 225.00
68881 20MHz Floating Point Coprocessor	\$ 345.00
MOTOROLA 68020 USERS MANUAL	\$ 18.00
MOTOROLA 68030 USERS MANUAL	\$ 18.00
MOTOROLA 68881 USERS MANUAL	\$ 18.00

## SBC ACCESSORY PACKAGE (M20-AP) \$1399.00

The package includes a PC-style cabinet with a custom backpanel, a 25 Megabyte (unformatted) hard disk and controller, a floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches, and all necessary internal cabling. (For use with SAB-90 serial connectors only.)

2nd 5" 80 FLOPPY & CABLES FOR M20-AP, ADD	\$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD	\$ 780.00
TO SUBSTITUTE 50MB HD FOR 25MB HD, ADD	\$ 290.00
TO SUBSTITUTE 80MB HD FOR 25MB HD, ADD	\$1500.00
TO SUBSTITUTE 155MB FOR 25MB HD, ADD	\$2100.00
60MB TEAC STREAMER WITH ONE TA	\$ 690.00
PKG. OF 5 TEAC TAPES	\$ 112.50

CUSTOM BACK PANEL PLATE (BPP-PC) \$ 44.00

## I/O EXPANSION BOARDS

### 16 PORT SERIAL BOARD ONLY (SBC-16S) \$ 335.00

The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

### RS232 ADAPTER (SAB-25, SAB-90 or SAB-8M) \$165.00

The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

### 60 LINE PARALLEL I/O BOARD (SBC-60P) \$398.00

The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

### PROTOTYPING BOARD (SBC-WW) \$75.00

The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual In-line Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

### I/O BUS ADAPTER (SBC-BA) \$195.00

The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

### ARCNET LAN board w/o Software (SBC-AN) \$475.00

The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNE modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN 120.00

## GMX MICRO-20 SOFTWARE

020 BUG UPDATE — PROMS & MANUAL \$150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD \$150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS. ALL SHIPPED STANDARD ON 5 1/4" DISKS 3 1/4" OPTIONAL IF SPECIFIED.

OS9/68020 PROFESSIONAL PAK \$850.00

Includes O.S., "C", uMACS EDITOR, ASSEMBLER, DEBUGGER, development utilities. 68881 support.

OS9/68020 PERSONAL PAK \$ 400.00

Personal OS-9 systems require a GMX Micro-20 development system running Professional OS-9/68020 for initial configuration.

BASIC (included in PERSONAL PAK) \$ 200.00

C COMPILER (included in PROFESSIONAL PAK) \$ 750.00

PASCAL COMPILER \$ 500.00

UniFLEX (for Micro-20) \$ 400.00

UniFLEX WITH REAL-TIME ENHANCEMENTS \$ 800.00

UniFLEX VM (for TWINGLE-20) \$ 600.00

UniFLEX VM REAL-TIME ENHANCEMENTS \$1000.00

## Other Software for UniFLEX

UniFLEX BASICW/PRECOMPILER \$ 300.00

UniFLEX C COMPILER \$ 350.00

UniFLEX COBOL COMPILER \$ 750.00

UniFLEX SCREEN EDITOR \$ 150.00

UniFLEX TEXT PROCESSOR \$ 200.00

UniFLEX SORT/MERGE PACKAGE \$ 200.00

UniFLEX VSAM MODULE \$ 100.00

UniFLEX UTILITIES PACKAGE \$ 200.00

UniFLEX PARTIAL SOURCE LICENSE \$1000.00

**GMX EXCLUSIVE VERSIONS, CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.**

ABSOFT FORTRAN (UniFLEX) \$1500.00

SCULPTOR (specify UniFLEX or OS9) \$ 995.00

FORTH (OS9) \$ 595.00

DYNACALC (specify UniFLEX or OS9) \$ 300.00

**GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.**

**ALL PRICES ARE F.O.B. CHICAGO IN U.S. FUNDS**

GMX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

**TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE.** Please allow 3 weeks for personal checks to clear. U.S. orders add \$5 handling if under \$200.00. Foreign orders add \$10 handling if order is under \$200.00. Foreign orders over \$200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

**CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS**

**GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.**

**GMX** 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

A Member of the CPI Family

# 68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

**Publisher**  
Don Williams Sr.

**Executive Editor**  
Larry Williams

**Production Manager**  
Tom Williams

**Office Manager**  
Joyce Williams

**Subscriptions**  
Stacy Power

## Contributing & Associate Editors

Ron Anderson	Dr. E.M. "Bud" Pass
Ron Voigts	Arl Weiler
Doug Lurie	Dr. Theo Elbert
Ed Law	& Hundreds More of Us

## Contents

"C" User Notes	7	Pass
Basically OS-9	17	Voigts
Software User Notes	21	Anderson
Logically Speaking	26	Jones
Mac-Watch	42	Anchors
FORTH	45	Lurie
Bit Bucket	53	
Classifieds	57	

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

## COMPUTER PUBLISHING, INC.

"Over a Decade of Service"



**68 MICRO JOURNAL**  
Computer Publishing Center  
5900 Cassandra Smith Road  
PO Box 849  
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

Copyrighted © 1987 by Computer Publishing, Inc.

68 Micro Journal is the *original* "DeskTop Publishing" product and has continuously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! We originated what has become traditional "DeskTop Publishing"! Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN, and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

## Subscription Rates

1 Year \$24.50 USA, Canada & Mexico \$34.00 a year.  
Others add \$12.00 a year surface, \$48.00 a year Airmail, USA funds. 2 years \$42.50, 3 years \$64.50 plus additional postage for each additional year.

## Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK-DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

*Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.*

## Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. We reserve the right to reject any letter or advertising material, for any reason we deem advisable. Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of \$15.50 for first 15 words. Add \$.60 per word thereafter. No classifieds accepted by telephone.



# OS-9 . . . Making Beautiful Music in the Key of "C"!

**W**hen you need to orchestrate beautiful music on your VME system, look to Microware for just the right score. Our finely tuned OS-9 Operating System is truly the maestro's choice when the project requires C language development. Our superlative C Compiler—also available in an optimized 68020 version—produces fast, compact ROMable code for your most demanding applications. A powerful Assembler, Linker and Symbolic Debugger assists in target development. And our C Compiler is source compatible with UNIX applications and available in cross-compiler configurations for Sun, VAX and Hewlett-Packard environments.

## **OS-9 Keeps On Performing Even After the Fat Lady Sings!**

Most operating systems hit a sour note when the project reaches completion. But not OS-9. Because of its modular design and UNIX-style architecture, your investment in OS-9 experience, tools and applications translates into a valuable resource for your company's future. OS-9 can be utilized again and again over your entire corporate product range.

## **An Accompaniment of Total Support**

Microware is proudly setting the industry's standard for customer support.

You'll find outstanding technical documentation that leaves nothing in doubt when it comes to real-world applications. A rigorous Quality Assurance program guarantees customer satisfaction by identifying trouble spots before they become customer problems. And with our

Customer Hotline, you are only a telephone call away from courteous and concise information. So join the growing legions of Microware "C" aficionados. Call us today and find out how you can create inspiring harmonies on your system.

*microware*<sup>®</sup>

MICROWARE SYSTEMS CORPORATION

1900 N.W. 114th Street  
Des Moines, IA 50322  
Phone 515-224-1929

Western Regional Office  
4401 Great America Parkway  
Santa Clara, CA 95054  
Phone 408-980-0201



Microware is a registered trademark of Microware Systems Corporation.  
OS-9/68020 is a trademark of Microware. VAX is a trademark of DEC.  
UNIX is a trademark of AT&T.

# MUSTANG-020 Super SBC™



**DATA-COMP Proudly Presents the First Under \$4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under \$2400 complete!**

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA.COMP "total package", consisting of a

**Data-Comp Division**

A Decade of Quality Service™

Systems World-Wide →

Computer Publishing, Inc. 5900 Cassandra Smith Road  
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

heavy duty metal cabinet, switching power supply with r/f/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under \$4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32 serial ports, at a cost of less than \$65 per port, in multiples of 8 port expansion options.

The SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less than \$2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.



*Available 12.5- 25 Mhz systems, call for special prices*

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3.45 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.



## Mustang-020 Mustang-08 Benchmarks

IBM AT 7300 Xenix Sys 3  
AT&T 7300 UNIX PC 68010  
DEC VAX 11/780 UNIX Berkeley 4.2  
DEC VAX 11/750  
68008 OS-9 68K 8 Mhz  
68000 OS-9 68K 10 Mhz  
MUSTANG-08 68008 OS-9 68K 10 Mhz  
MUSTANG-020 68020 OS-9 68K 16 Mhz  
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz

32 bit Integer	Register Long
9.7	
7.2	4.3
3.6	3.2
5.1	3.2
18.0	9.0
6.5	4.0
9.8	6.3
2.2	0.88
1.8	1.22

Main()

register long i;  
for (i=0; i < 999999; i++)

Estimated MIPS - MUSTANG-020 ... 4.5 MIPS,  
Burst to 8 - 10 MIPS; Motorola Speed

### OS-9

OS-9 Professional Ver	\$850.00
*Includes C Compiler	
Basic 09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: \$100.00)	100.00
Parvus 77	750.00
Micro as Pascal	500.00
Onesight Pascal	900.00
Style Graph	495.00
Style Spell	195.00
Style Merge	175.00
Style Graph Spell Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Scalptor* (see below)	995.00
COM	125.00

### UniFLEX

UniFLEX (68020 ver)	\$450.00
Screen Editor	150.00
Sort Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
MODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Parvus 77	450.00
Scalptor* (see below)	995.00

Standard MUSTANG-020 <sup>TM</sup> shipped 12.5 Mhz.	
add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/68881	750.00

16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below RS232 Adapter	165.00

Each card supports 4 additional ser. ports  
(total of 36 serial ports supported)

60 line Parallel I/O card	398.00
Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	

Prototype Board	75.00
area for both dip and POA devices at a pre-wired memory area up to 512K DRAM.	

SBC-AN	475.00
Interface between the system and ARCNET modified token-passing LAN. (this option optional - call LAN software drivers)	120.00

Expansion for Motorola I/O Channel Modules	\$195.00
Special for complete MUSTANG-020 <sup>TM</sup> system buyers - Scalptor*	\$695.00. SAVE \$300.00
Software Discounts	

All MUSTANG-020<sup>TM</sup> system and board buyers are entitled to  
discounts on all listed software: 10-70% depending on item. Call or  
write for details. Discounts apply after the sale as well.

## Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path  
32-bit wide data and address buses, non-multiplexed  
on chip instruction cache  
object code compatible with all 68XXX family processors  
enhanced instruction set - math co-processor interface  
68881 math hi-speed floating point co-processor (optional)  
direct extension of full 68020 instruction set  
full support IEEE 754, draft 10.0  
transcendental and other scientific math functions  
2 Megabyte of SIP RAM (512 x 32 bit organization)  
up to 256K bytes of EPROM (64 x 32 bits)  
4 Asynchronous serial I/O ports standard  
optional to 20 serial ports  
standard RS-232 interface  
optional network interface  
Tered 8 bit parallel port (1/2 MC68230)  
Centronics type pinout  
expansion connector for I/O devices  
16 bit data path  
256 byte address space  
2 interrupt inputs  
clock and control signals  
Motorola I/O Channel Modules  
time of day clock/calendar w/battery backup  
controller for 2, 5 1/4" floppy disk drives  
single or double side, single or double density  
35 to 80 track selectable (48-96 TPI)  
SASI interface  
programmable periodic interrupt generator  
interrupt rate from micro-seconds to seconds  
highly accurate time base (5 PPM)  
5 bit sense switch, readable by the CPU  
Hardware single-step capability



Don't be misled!  
ONLY Data-Comp  
delivers the Super  
MUSTANG-020

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission,  
Government Agencies as well as Universities, Business, Labs. and other Critical Applications  
Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level  
V compatibility and low cost is a must.

The  
P  
R  
O  
!

Only the "PRO" Version  
of OS-9 Supported!



This is **HEAVY DUTY**  
Country!

For a limited time we will offer a \$400  
trade-in on your old 68XXX SBC.  
Must be working properly and  
complete with all software, cables and  
documentation.  
Call for more information

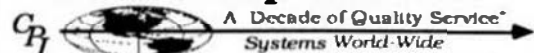
### Price List:

Mustang-020 SBC	\$2490.00
Cabinet w/switching PS	\$299.95
5"-80 track floppy DS/DD	\$269.95
Floppy Cable	\$39.95
OS-9 68K Professional Version	\$850.00
C Compiler (\$500 Value)	NAC
Winchester Cable	\$39.95
Winchester Drive 25 Mbyte	\$895.00
Hard Disk Controller	\$395.00
Shipping USA UPS	\$20.00
UniFLEX	Less \$100.00
MC68881 f/p math processor	Add \$275.00
16.67 Mhz MC68020	\$375.00
16.67 Mhz MC68881	\$375.00
20 Mhz MC68020 Sys	\$750.00
Note all 68881 chips work with 20 Mhz Sys	
Total:	\$5299.80

**Save \$1000.00**  
**Complete**  
**25 Mbyte HD System**  
**\$4299.80**  
**85Mbyte HD System**  
**\$5748.80**

Note: Only Professional OS-9 Now Available (68020 Version)  
Includes (\$500) C Compiler - 68020 & 68881 Supported -  
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

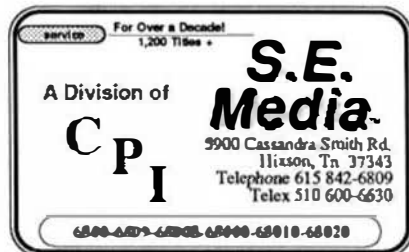
## Data-Comp Division



Computer Publishing, Inc. 5900 Cassandra Smith Road  
Telephone 615 842-4801 - Telex 510 600-6630 Hixson, TN 37343

# PAT - JUST

**PAT**  
With 'C' Source  
**\$229.00**



**PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR** with all the best of PIE. For those who swore by and loved PIE, this is for YOU! All PIE features & much more! Too many features to list. And if you don't like ours, change or add your own. C source included. Easily configured to your CRT terminal, with special configuration section. No sweat!

**68008 - 68000 - 68010 - 68020 OS-9 68K \$229.00**

## COMBO PAT/JUST

### Special \$249.00

### JUST

**JUST from S. E. MEDIA - -** Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with PAT or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

**68008 - 68000 - 68010 - 68020 OS-9 68K**  
**With 'C' source \$79.95**





*The C Programmers  
Reference Source.  
Always Right On Target!*

## C User Notes

### A Tutorial Series

By: Dr. E. M. 'Bud' Pass  
1454 Latta Lane N.W.  
Conyers, GA 30207  
404 483-1717/4570

*Computer Systems Consultants*

### INTRODUCTION

This chapter discusses the use of benchmarks for testing the performance of selected computer systems. Benchmarking computers is a difficult task because of the large number of parameters which must be evaluated when determining the performance of a given system. It should be a necessary part of the development and selection of new computer and software systems to ensure that the performance is as expected.

### BENCHMARKS

Since benchmarking is simultaneously important and difficult, much time and effort has been spent on analyzing the problem, but little time and effort is usually spent on properly applying benchmarking techniques in practical situations.

The infamous sieve program has often been used in recent years as a means of comparing the performance of a limited number of C compilers in particular cases. Unfortunately, it has been used in many published

studies to compare complete processors and computer systems, with the original assumptions forgotten, if they were ever known.

Benchmarks are normally developed and used to compare performance characteristics in one or more of the following classes, arranged in increasing order of difficulty of establishment and interpretation: *one compiler before and after modification on the same computer and operating system, one compiler with different operating system on the same computer, one compiler with different hardware configuration or with different operating systems, multiple compilers on the same operating system on the same computer, multiple compilers on the same computer with different operating systems, multiple compilers on different operating system on the same computer.*

The SVID test suite, available from several sources, is an example of an attempt to provide a thorough test of a particular operating system on a particular hardware

configuration. It is certainly a benchmark for certification of an operating system, including c compiler, assembler, linkage editor, low-level interfaces, etc.

In all cases, the benchmarks should all be checked for time and accuracy of execution. Many benchmarks attempt to carefully determine the speed of their execution, but do not attempt to determine if they have been executed correctly. Both internal and external checks should be used. Internal checks compute the same results in two very different manners or compare results to pre-computed values and indicate success or failure (or failure only). External checks involve comparison of results output by the benchmark program with pre-computed values, but by other programs or by humans. Internal checks may be inadequate in certain failure modes.

The first case, comparing the same compiler before and after modification, is the most straightforward in application, and illustrates several of the characteristics which must be considered in the more complex cases. The benchmark program or programs should be designed to test not only the specific areas of concern, but to test other areas to ensure that new problems have not been introduced. The areas should be tested separately. One really important issue is to ensure that the actions of the benchmark program are understood, and that relevant parameters really are being accurately measured. If the benchmark supports several major options, the benchmark should be run with each relevant option.

The other cases are progressively more complex, but should be based upon the same principles as the simpler cases. The primary guidelines should be to understand how the program functions and to ensure that it functions as expected in measuring the relevant situational parameters. The duration of the tests must be long enough to ensure that the effect of transient variations is minimized. Normally, benchmarks on multi-user systems should be performed several times, with minimal system loading.

## BENCHMARK FAILURES

It is not difficult to find published examples of benchmark programs which were claimed to measure one parameter but, in fact, measured some other parameter, none in particular, or provided misleading results.

The sieve benchmark program allows the comparison of so little of a system's capabilities that it must be logically discarded as a valid benchmark, despite its use in hundreds of articles and advertisements. It also provides no assurance that it has performed its operations correctly.

A published floating-point benchmark program actually compares transcendental floating-point function performance. Systems providing the best ranking were determined to have used special floating-point hardware. Systems providing the next-best ranking were determined to have transcendental functions coded in assembler language. Systems providing the lower rankings were determined to have transcendental functions coded in the C language. Therefore, the benchmark program provided no or misleading information about floating-point speed in the C language!

Another published benchmark program supposedly provided a means of ranking the speed of pointer operations. The program actually consisted primarily of a doubly-nested for loop surrounding two simple pointer operations, and the program did not account for the overhead

due to the nested for loops, which was later determined to be far higher than the time required to perform the pointer operations.

An advertisement from a C compiler manufacturer claimed that their compiler performed the sieve benchmark several times faster than their best competitor's C compiler. When the compilers were benchmarked by impartial observers, the opposite conclusion was drawn. The manufacturer claiming to have the faster compiler had apparently run only one iteration of the sieve benchmark through their compiler, while the competitor's benchmark had run ten iterations! As David Horowitz says, "Read Those Labels".

## DHRYSTONE BENCHMARK

The Dhrystone benchmark program was originated by Reinhold P. Weicker, in the Communications of the ACM Volume 27, Number 10, October 1984, page 1013. It was translated to the C language by Rick Richardson (lhnp4!castor!pcrat!rick).

In this program, the keywords "if", "for", and "while" account for over 70 percent of the control statements. It uses data types char, int, long, but not float or double. In execution counts, 53 percent of the statements are assignments, 32 percent are control statements, and 15 percent are function calls.

Dhrystone provides defines for C compilers which don't support enums, and for those which do not support



structure assignment. It also provides defines for those C compilers which support the time or times functions, or neither, and for variations in the C library implementations of these functions. It also supports a major option

REG=register to select or deselect compiler register usage.

This benchmark suffers from the problem that it does not compute and output any values which may be used to

check that it has performed its operations correctly. Its only output is the Dhrystone rating.

### BENCHMARK RESULTS

The following results have been noted, rounded to the nearest 25:

Computer	CPU	OS	Dhrystones/sec.
SSB Chieftain	6809-2	OS-9 1.2	225
FORTUNE 16:32	68000-4	UNIX FORPRO	300
PC-XT clone	8088-5	MSDOS 2.1	325
PT 68000	68000-10	SKDOS	510
PC-XT clone	V-20-8	MSDOS 3.1	675
AT&T 3B2/300	WE32000-7.2	UNIX 5.2	675
PC-XT clone	8088-10	MSDOS 3.1	700
PC-XT clone	V-20-10	MSDOS 3.1	800
DEC 11/750	VAX	ULTRIX 1.2	850
CTI Mini-Frame	68010-10	CTIX V.2	950
Sun 2/120	68010-10	SUN 2.2	1000
AT&T 3B2/400	WE32000-10	UNIX 5.2	1025
Sequent 8000	32032-10	DYNIX	1100
GMX Micro-20	68020-12	OS-9 1.2	1250
DEC MICROVAX II	VAX	VMS 4.5	1325
Compaq II	80286-8	MSDOS 3.1	1350
DEC MICROVAX II	VAX	ULTRIX 1.2	1375
DEC 11/780	VAX	ULTRIX 1.2	1450
AT&T 3B15	WE32100-14	UNIX 5.2.1	1800
DEC 11/785	VAX	ULTRIX 1.2	1800
PC-AT clone	80286-10	MSDOS 3.1-MSOFTC	1825
PC-AT clone	80286-10	UNIX 5.2	1875
PC-AT clone	80286-10	MSDOS 3.1-TURBOC	2000
DEC 11/785	VAX	VMS 4.3	2000
Sun 3/50	68020-15	SUN 3.0	2300
AT&T 3B2/600	WE32100-16	UNIX 5.3.1	2700
Compaq 386	80386-16	XENIX 5.2	2750
Arete 1200	68020-12	UNIX V.2.2	2800
Pyramid 90x	DCU-8	UNIX V.2	2900
Sun 3/160	68020-17	SUN 3.0	2950
Tandem LXN 2	68020-17	UNIX V.2	3000
Tower 32/600	68020-16	UNIX V.1.02	3325
Tower 32/800	68020-16	UNIX V.1.00	3325
Compaq 386	80386-20	XENIX 5.2	3425
Pyramid 98x	DCU-10	UNIX V.2	3600
HP 9000/350	68020-25	HPUX A.B1	5925
Intel 386/24	80386-20	UNIX V.3	6000
HP 9000/840	SPECTRUM	HPUX A.B1	6675

*Editors Note: The GMX system and the DATA-COMP MUSTANG-020, running 20 Mhz clocks in at over 2000.*

The Dhrystone benchmark is a standard comparison of C compiler and computer performance, not a test of overall system throughput.

When several tests were run, with differing results, the lowest value noted was used. The results may also be somewhat inaccurate due to variable system loading.

### EXAMPLE C PROGRAM

Following is this month's example C program: it is the Dhrystone program, described in the text above.

```
/*
 *
 *      "DHRYSTONE" Benchmark Program
 *
 *      The following program contains statements of a high-level programming
 *      language (C) in a distribution considered representative:
 *
 *      assignments          53%
 *      control statements    32%
 *      procedure, function calls  15%
 *
 *      100 statements are dynamically executed. The program is balanced with
 *      respect to the three aspects:
 *          - statement type
 *          - operand type (for simple data types)
 *          - operand access
 *              operand global, local, parameter, or constant.
 *
 *      The combination of these three aspects is balanced only approximately.
 *
 *      The program does not compute anything meaningful, but it is
 *      syntactically and semantically correct.
 */

/* Accuracy of timings controlled by next two lines */
#define LOOPS 50000 /* Use this for slow or 16 bit machines */
/*#define LOOPS 500000 /* Use this for faster machines */

/* Compiler dependent options */
#undef ENUM /* Define if compiler has enum's */
#undef STRUCTASSIGN /* Define if compiler can assign structures */

/* define only one of the next two defines */
#define TIME /* Use time(2) time function */
/*#define TIMES*/ /* Use times(2) time function */

/* define the granularity of your times(2) function (when used) */
/*#define HZ 50 /* times(2) returns 1/50 second (europe?) */
#ifndef ATT3B
#define HZ 60 /* times(2) returns 1/60 second (most) */
#else
#define HZ 100 /* times(2) returns 1/100 second (WECO) */
#endif

char Version[] = "1.1";

#ifndef STRUCTASSIGN
#define structassign(d, s) memcpy(&(d), &(s), sizeof(d))
#else
#define structassign(d, s) d = s
#endif
```

```

#ifdef ENUM
typedef enum
    {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
#else
#define Ident1 1
#define Ident2 2
#define Ident3 3
#define Ident4 4
#define Ident5 5
#define Enumeration int
#endif

#define OneToThirty int
#define OneToFifty int
#define CapitalLetter char

struct Record
{
    struct Record      *PtrComp;
    Enumeration        Discr;
    Enumeration        EnumComp;
    OneToFifty         IntComp;
    char               StringComp[30];
};

#define RecordType      struct Record
#define RecordPtr       RecordType *
#define boolean         int

#define NULL            0
#define TRUE            1
#define FALSE           0

#ifdef REG
#define REG
#endif

extern Enumeration      Func1();
extern boolean          Func2();

#ifdef TIMES
#include <sys/types.h>
#include <sys/times.h>
#endif

main()
{
    Proc0();
    exit(0);
}

/*
 * Package 1
 */
int      IntGlob;
boolean  BoolGlob;
char     Char1Glob;
char     Char2Glob;
int      Array1Glob[51];
int      Array2Glob[51][51];
RecordPtr PtrGlb;
RecordPtr PtrGlbNext;

```

```

Proc0()
{
    OneToFifty          IntLoc1;
    REG OneToFifty       IntLoc2;
    OneToFifty           IntLoc3;
    REG char              CharLoc;
    REG char              CharIndex;
    Enumeration           EnumLoc;
    char                  String1Loc[30];
    char                  String2Loc[30];
    extern char           *malloc();
    register unsigned int i;

#ifdef TIME
    long                  time();
    long                  starttime;
    long                  benchtime;
    long                  nulltime;

    starttime = time( (long *) 0);
    for (i = 0; i < LOOPS; ++i);
    /* Computes overhead of looping */
    nulltime = time( (long *) 0) - starttime;
#endif
#ifdef TIMES
    time_t                starttime;
    time_t                benchtime;
    time_t                nulltime;
    struct tms            tms;

    times(&tms); starttime = tms.tms_utime;
    for (i = 0; i < LOOPS; ++i);
    times(&tms);
    /* Computes overhead of looping */
    nulltime = tms.tms_utime - starttime;
#endif

    PtrGlbNext = (RecordPtr) malloc(sizeof(RecordType));
    PtrGlb = (RecordPtr) malloc(sizeof(RecordType));
    PtrGlb->PtrComp = PtrGlbNext;
    PtrGlb->Discr = Ident1;
    PtrGlb->EnumComp = Ident3;
    PtrGlb->IntComp = 40;
    strcpy(PtrGlb->StringComp,
           "DHRYSTONE PROGRAM, SOME STRING");
    strcpy(String1Loc, "DHRYSTONE PROGRAM, 1'ST STRING");
    Array2Glob[8][7] = 10;

/*****
- Start Timer -
*****/
#ifdef TIME
    starttime = time( (long *) 0);
#endif
#ifdef TIMES
    times(&tms); starttime = tms.tms_utime;
#endif
    for (i = 0; i < LOOPS; ++i)
    {
        Proc5();
        Proc4();
        IntLoc1 = 2;
        IntLoc2 = 3;

```



```

strcpy(String2Loc,
        "DHRYSTONE PROGRAM, 2'ND STRING");
EnumLoc = Ident2;
BoolGlob = ! Func2(String1Loc, String2Loc);
while (IntLoc1 < IntLoc2)
{
    IntLoc3 = 5 * IntLoc1 - IntLoc2;
    Proc7(IntLoc1, IntLoc2, &IntLoc3);
    ++IntLoc1;
}
Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
Procl(PtrGlb);
for (CharIndex = 'A'; CharIndex <= Char2Glob;
    ++CharIndex)
    if (EnumLoc == Funcl(CharIndex, 'C'))
        Proc6(Ident1, &EnumLoc);
IntLoc3 = IntLoc2 * IntLoc1;
IntLoc2 = IntLoc3 / IntLoc1;
IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
Proc2(&IntLoc1);
}

/*****
- Stop Timer -
*****/

#ifdef TIME
    benchtime = time( (long *) 0) - starttime - nulltime;
    printf("Dhrystone(%s) time for %ld passes = %ld\n",
        Version, (long) LOOPS, benchtime);
    printf("This machine benchmarks at %ld dhrystones/second\n",
        ((long) LOOPS) / benchtime);
#endif
#ifdef TIMES
    times(&tms);
    benchtime = tms.tms_utime - starttime - nulltime;
    printf("Dhrystone(%s) time for %ld passes = %ld\n",
        Version,
        (long) LOOPS, benchtime/HZ);
    printf("This machine benchmarks at %ld dhrystones/second\n",
        ((long) LOOPS) * HZ / benchtime);
#endif
}

Procl(PtrParIn)
REG RecordPtr PtrParIn;
{
#define NextRecord    (*(PtrParIn->PtrComp))

    structassign(NextRecord, *PtrGlb);
    PtrParIn->IntComp = 5;
    NextRecord.IntComp = PtrParIn->IntComp;
    NextRecord.PtrComp = PtrParIn->PtrComp;
    Proc3(NextRecord.PtrComp);
    if (NextRecord.Discr == Ident1)
    {
        NextRecord.IntComp = 6;
        Proc6(PtrParIn->EnumComp,
            &NextRecord.EnumComp);
        NextRecord.PtrComp = PtrGlb->PtrComp;
        Proc7(NextRecord.IntComp, 10,
            &NextRecord.IntComp);
    }
}

```

```

    }
    else
        structassign(*PtrParIn, NextRecord);

#undef NextRecord
}

Proc2(IntParIO)
OneToFifty    *IntParIO;
{
    REG OneToFifty    IntLoc;
    REG Enumeration    EnumLoc;

    IntLoc = *IntParIO + 10;
    for(;;)
    {
        if (Char1Glob == 'A')
        {
            -IntLoc;
            *IntParIO = IntLoc - IntGlob;
            EnumLoc = Ident1;
        }
        if (EnumLoc == Ident1)
            break;
    }
}

Proc3(PtrParOut)
RecordPtr    *PtrParOut;
{
    if (PtrGlob != NULL)
        *PtrParOut = PtrGlob->PtrComp;
    else
        IntGlob = 100;
    Proc7(10, IntGlob, &PtrGlob->IntComp);
}

Proc4()
{
    REG boolean    BoolLoc;

    BoolLoc = Char1Glob == 'A';
    BoolLoc |= BoolGlob;
    Char2Glob = 'B';
}

Proc5()
{
    Char1Glob = 'A';
    BoolGlob = FALSE;
}

extern boolean Func3();

Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
{
    *EnumParOut = EnumParIn;
    if (! Func3(EnumParIn) )
        *EnumParOut = Ident4;
    switch (EnumParIn)
    {
        case Ident1:    *EnumParOut = Ident1; break;
    }
}

```

```

        case Ident2:    if (IntGlob > 100)
                        *EnumParOut = Ident1;
                        else
                        *EnumParOut = Ident4;
                        break;
        case Ident3:    *EnumParOut = Ident2; break;
        case Ident4:    break;
        case Ident5:    *EnumParOut = Ident3;
    }
}

Proc7(IntParI1, IntParI2, IntParOut)
OneToFifty    IntParI1;
OneToFifty    IntParI2;
OneToFifty    *IntParOut;
{
    REG OneToFifty IntLoc;

    IntLoc = IntParI1 + 2;
    *IntParOut = IntParI2 + IntLoc;
}

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
int            Array1Par[51];
int            Array2Par[51][51];
OneToFifty    IntParI1;
OneToFifty    IntParI2;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;

    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1);
        ++IntIndex)
        Array2Par[IntLoc][IntIndex] = IntLoc;
    ++Array2Par[IntLoc][IntLoc-1];
    Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    IntGlob = 5;
}

Enumeration Func1(CharPar1, CharPar2)
CapitalLetter CharPar1;
CapitalLetter CharPar2;
{
    REG CapitalLetter CharLoc1;
    REG CapitalLetter CharLoc2;

    CharLoc1 = CharPar1;
    CharLoc2 = CharLoc1;
    if (CharLoc2 != CharPar2)
        return (Ident1);
    else
        return (Ident2);
}

boolean Func2(StrParI1, StrParI2)
char    StrParI1[30];
char    StrParI2[30];
{
    REG OneToThirty IntLoc;

```

```

    REG CapitalLetter      CharLoc;

    IntLoc = 1;
    while (IntLoc <= 1)
        if (Func1(StrParI1[IntLoc],
                  StrParI2[IntLoc+1]) == Ident1)
        {
            CharLoc = 'A';
            ++IntLoc;
        }
    if (CharLoc >= 'W' && CharLoc <= 'Z')
        IntLoc = 7;
    if (CharLoc == 'X')
        return (TRUE);
    else
    {
        if (strcmp(StrParI1, StrParI2) > 0)
        {
            IntLoc += 7;
            return (TRUE);
        }
        else
            return (FALSE);
    }
}

boolean Func3(EnumParIn)
REG Enumeration EnumParIn;
{
    REG Enumeration EnumLoc;

    EnumLoc = EnumParIn;
    if (EnumLoc == Ident3) return (TRUE);
    return (FALSE);
}

#ifdef STRUCTASSIGN
memcpy(d, s, l)
register char *d;
register char *s;
register int l;
{
    while (l--) *d++ = *s++;
}
#endif
EOF

```

*FOR THOSE WHO NEED TO KNOW*

**68 MICRO  
JOURNAL™**



# Basically OS-9

Dedicated to the serious OS-9 user.  
The fastest growing users group world-wide!  
6809 - 68020

*A Tutorial Series*

By: Ron Voigts  
2024 Baldwin Court  
Glendale Heights, IL

## WHERE DID ALL THE COMMANDS COME FROM?

I came across an interesting problem the other day. I was looking at the OS-9 Level II system for the Color Computer. Now the color computer comes with 128 K of RAM. Once Level II is booted it has about 56K left. This brings back an earlier issue with the Color Computer and Level I. It is the old problem of not enough memory. Actually there is somewhat more memory available. If I remember right, running Level I had about 46K left. It is a sufficient amount to run things, but it does not leave a large margin. With the Coco 3, windows are also available. Use a couple of these and the memory goes away quite fast.

Back to the problem. While examining memory with MDIR, I noticed the most often used commands were pre-loaded. Ah-hah! I thought. I can free up some extra memory by unlinking. I used UNLINK on them and then tried MFREE again. I still only had 56K of RAM left. Quickly I keyed in MDIR and all the commands were still there. They must be part of the system I guessed.

I did an IDENT on the OS9Boot file. Swiftly all the modules it contained flew by. Not one of the commands was in there. Now this was becoming a mystery. Where were those commands coming from?

I went to bed that night thinking about where the commands were coming from. I kept telling myself that the basics of OS-9

had not been rewritten. Then it came to me. I knew the answer. In my younger days, I would have jumped out bed and rushed to the computer see if I was right. However, I decided to wait until morning and try out my hunch.

The next morning I did another IDENT on OS9Boot. Sure enough it was there. Or should I say, not there! The SHELL was not part of the boot. I did a quick IDENT on the SHELL which is in the commands directory. And there was the missing information. The SHELL was the first command and it was followed by all the OS-9 commands that I found in memory. Whenever, I booted up, CC3GO ( this is the Color Computer version SYSGO ) would fork to SHELL. Not seeing it in memory, it would search the execution directory and load it. SHELL's link count would be increased by one. The other modules would be along for the ride. As long as the SHELL is linked, the other modules stay in memory.

I explained the principle a few columns ago, but it will not hurt to review it. Memory is allocated in large chunks in a Level II system. Usually they are 2K in size. But this can vary. Another size might be 4K. The Color Computer Level II is 8K. If the SHELL is loaded into memory by itself, the rest of the 2K ( or whatever ) of memory is left empty. For a shell that is 1K long, only half of the memory is used. The other half has been allocated, but lays idle.

It is not entirely a great sin as I have made it out to be to waste the memory space. In fact in Level II since memory is allocated this way, it is inevitable that memory will be wasted. This is the sacrifice that must be made for an efficient memory management system. Every time a program is executed, there will be some waste. In a Level I system this can be minimal compared to the Level II system. In Level I, memory is allocated in 256 byte chunks called pages. A module loaded into memory is assigned to the start of a page. But all the modules reside in the same memory space.

An example can help to clarify this. In my Level I system, STARTUP has the following in it:

```
LOAD DIR DEL RENAME
```

If I were to use MDIR E after booting we would see something like this.

```
Module directory at 09:02:56
Addr Size Typ Rev Attr Use Module name
-----
E871 4FA 11 1 r... 1 Shell
A100 24D 11 1 r... 1 Mdir
A700 35D 11 1 r... 1 Dir
A600 A5 11 1 r... 1 Del
A400 11D 11 1 r... 1 Rename
```

( This is an abbreviated listing, showing only a few entries. ) Notice DIR is \$24D bytes long and is positioned at \$A100, DEL is \$AD long and is at \$A600 and RENAME is \$11D long and is at \$A400. These three modules are collectively \$51F or 1311 bytes of memory long. They use \$700 or 1792 bytes of available memory. This means that 73% of the memory is actually being used.

If I add the same thing to my Level II STARTUP file and do an MDIR E, it would look like this.

```
Module Directory at 14:54:06
Block Offset Size Typ Rev Attr Use Module Name
-----
7 0 529 11 2 r... 2 Shell
1C 0 26A 11 1 r... 1 MDir
15 0 199 11 1 r... 1 Del
16 0 2A2 11 1 r... 1 Dir
17 0 11D 11 1 r... 1 Rename
```

( Again this is an abbreviated listing. )

Here DIR, DEL and RENAME are assigned to a different block of memory. Each block is 2K bytes long. DIR is \$2A2 bytes long at block \$16, DEL is \$199 at \$15 and RENAME is \$11D at \$17. Together they occupy \$4D8 or 1240 bytes. The remainder of the 6K or 6144 bytes is unused. This means only 20% of the memory is being used.

If the system is large enough, there is not too much concern over whether a few K of memory is wasted. I run 1 MEG of memory on the my regular Level II system. I can afford to waste a little memory here and there. But if memory is a constraint, then it might be necessary to consider ways to get the most out of it. One way is to put all the commonly used modules into one file.

The advantage to putting them all together is that they will be loaded into one area of memory. What happens is that when a name is forked to and it is not in memory, the entire file is loaded. No matter what is in the file as long as its CRCs and header parities are good, it is loaded. The modules are loaded into one continuous area. The first module is linked. As long as its link count ( or any other of the modules in that area ) is not 0, the memory allocation is good. This is where the convenience of putting the shell first and following it with the commonly used commands is handy. The shell will probably always be running. So everything else will stay with it.

Even with a lot of memory in my Level II, 1 MEG system, I have started to merge the commonly used commands. I found the easiest way to do it was with a little procedure file. Here is the one that I use.

```
chd /d0/cmds
merge shell copy del dir display ident >shell.1
merge echo link list load unlink save >shell.2
merge pwd pxd free mfree format attr rename >shell.3
rename shell shell.old
merge shell.1 shell.2 shell.3 >shell
del shell.1 shell.2 shell.3
attr shell e pe
```

This little procedure changes the working directory to the commands directory. It merges the files that I want to reside in memory. It renames the old SHELL to SHELL.OLD. The intermediate names are then merged to the file name shell and they are deleted. Finally the new SHELL's attributes are changed to make the file executable. I did not delete the old shell. Rather I leave it behind renamed. My belief that it will be available for future use.

The shell for this is longer than one block of memory. But the entire memory space that it occupies is much smaller than it would be if I had loaded these separately. You have to consider how much memory is available in your system. If you are running Level II on a small amount of memory, you might only want to make a file called shell that will fill only about one block in memory. This would be the minimal file size. If your system is larger and you don't mind using blocks, you can expand this to use more blocks of memory. You will have to play around with this to see what the best size would be.

#### BASIC09 TOOLS FOR LEVEL II

This same principle is applicable to the modules in used in other applications. BASIC09 TOOLS is a good example to try this on. This is a software package that is available from Southeast Media. I authored it. It contains 21 subroutine modules for use with BASIC09. If the modules are loaded into memory in Level II with each one occupying 2K of memory, they will use up 41K of available memory. Listing 1 presents a program that will alleviate this problem.

The program in Listing 1 will do nothing really important except act as a header module for a larger file of merged modules from the Basic09 Tools. The module does nothing. If it should be run, it will return without error. To use this module with the Tools the following should be created

in a procedure file. Before running this assemble the code in Listing 1 to a module called tools.obj.

```
chd /dl/cmds
merge tools.obj cfill dpeek dpoke fpos >tools.1
merge fsize ftrim getpr getopt getusr >tools.2
merge gtime upper insert lower >tools.3
merge ready setprior syscall setusr >tools.4
merge setopt stime space swap >tools.5
merge tools.1 tools.2 tools.3 tools.4 tools.5 >tools
del tools.1 tools.2 tools.3 tools.4 tools.5
attr tools e pe
```

Running this little file will create a file called tools in the commands directory on drive D1. When you want to use the Tools in Level II, enter:

OS9: load tools

The entire file will be loaded with all the modules. The first module, tools will be linked. This will insure that all the modules will remain in memory. And what is more important they will reside in one area of memory. When finished you can remove them with UNLINK. Enter:

OS9: unlink tools  
and they all will be gone.

I have considered that there are some OS9 users who do not have an assembler or really want to get involved with assembly language. So in Listing 2 I have included a Basic09 program that will generate the module tools.obj. If you only have Basic09, key in this program and it will create the module. Then use the above procedure to create the merged files.

That is it for this month. Have a good month. See you next time.

#### LISTING 1

```
00001      *
00002      *
00003      * NAME: TOOLS  for Basic09
00004      * FILE: TOOLS.A
00005      * BY: Ron Voigts
00006      * DATE: 19-Jul-86
00007      *
00008      *
00009      *
00010      * USAGE:
00011      *      none
00012      *
00013      *
00014      *
00015      * FUNCTION:
```

```

00016      *      This program is to be merged with
00017      *      the other Basic09 Tools. It is
00018      *      the first module in the package.
00019      *
00020      .....
00021      *
00022      use /d0/defs/
00023      defsf file
00023      ifpl
00032      endc
00033
00034      *
00035      0021      TYPE      set      SBRTN+OBJECT
00036      0081      REVS      set      REENT+1
00037      0000 87CD0017      mod
00037      TLEND, TLNAME, TYPE, REVS, ENTRY, 0
00038      000D 546F6F6C      TLNAME      fcs      "Tools"
00039
00040      *
00041      0012      ENTRY      equ      *
00042      * This does nothing, but returns without
error
00043      0012 5F      clrb
00044      0013 39      rts
00045      0014 809A4D      emod
00046      0017      TLEND      EQU      *
00047      end

00000 error(s)
00000 warning(s)
$0017 00023 program bytes generated
$0000 00000 data bytes allocated
$264A 09802 bytes used for symbols

```

#### LISTING 2

```

PROCEDURE tools_make
0000      (* .....
0017      (*
001A      (* Name: make_tools
002D      (* Date: 21-DEC-87
003F      (* By: Ron Voigts
0050      (* Language: BASIC09
0064      (*
0067      (* .....
007E      (*
0081      (* Function:
008D      (* This program will create
00A8      (* an executable file called
00C4      (* "tools.obj" that can be
00DE      (* merged with the other
00F6      (* Basic09 Tools.
0107      (*
010A      (* .....

```

```

0121
0122      (* Dimension variables used
013D      DIM s:BYTE
0144      DIM path:BYTE
014B      DIM count:INTEGER
0152
0153      (* Set up the count for reading
0172      count:=24
0179
017A      (* Open the file
018A      CREATE #path,"tools.obj":WRITE
019E
019F      (* Loop for COUNT times
01B6      (* Reading data and writing it
01D3      WHILE count>0 DO
01DF          READ s
01E4          PUT #path,s
01EE          count:=count-1
01F9      ENDWHILE
01FD
01FE      (* Close the path
020F      CLOSE #path
0215
0216      (* Fix the attributes
0228      SHELL "attr tools.obj pe e"
0242
0243      END
0245
0246      (* The data to be written
025F      DATA $87,$CD,$00,$17,$00
0277      DATA $0D,$21,$81,$0F,$00
028F      DATA $12,$00,$00,$54,$6F
02A7      DATA $6F,$6C,$F3,$5F,$39
02BF      DATA $B0,$9A,$4D

```

EOF

**FOR THOSE WHO NEED TO KNOW**

**68 MICRO  
JOURNAL™**



# SOFTWARE

---

## A Tutorial Series

By: Ronald W. Anderson  
3540 Sturbridge Court  
Ann Arbor, MI 48105

# USER

---

*From Basic Assembler to HLL's*

# NOTES

This is not a criticism of anyone, just an observation of fact. Similar occurrences seem to happen all too frequently with regard to computers and computer operating systems. The present subject is the Mustang-08A system sold by Data Comp, which uses the Peripheral Technology single board 68008 based computer. The fact that this system is put together from parts from several different sources helps greatly to compound the confusion.

First of all, the Western Digital hard disk controller can control up to three hard disk drives, which Western Digital calls 1, 2, and 3. Peripheral Technology found that the controller defaults to 1 on power up and/or power down, and that momentary power interruptions frequently caused crashes on what WD calls drive 1. They therefore chose to leave drive 1 unconnected and use drives 2 and 3 (good thinking there). Microware calls the hard disk drive devices /h0 and /h1. WD drive 2 is therefore /h0, and drive 3 is /h1. SK-DOS being very FLEXlike numbers drives 0,1,2,3, etc. It allows the user to configure any mix of hard and floppy disk drives as sequential logical drive numbers in any order. The configuration utility calls the hard drives h0, h1, and the floppy drives f0, f1, etc. The syntax for defining Logical Drive 0 to be f1, for example is LO=F1. After assigning the drive numbers, they are referred to as 0, 1, 2, etc.

If you are not thoroughly confused yet, the monitor in the Mustang-08 A, when you choose the option to park the hard disk, asks you "A or B?" A corresponds to /h0 and B to /h1. Since the monitor for this machine was certainly not done for an IBM compatible, I wonder why suddenly the drives are referred to as A and B?

This little confusion is mild compared to some of the others that have arisen primarily because of "tradition" in computing. If you have ever tried to use modules such as A/D converters and the like that are "computer compatible" or struggled over printer manuals, you probably wonder as I do, why something as simple as numbering the bits in an n bit word can't be standardized. Motorola, in their 8 bit processor line at least, has always numbered their data lines B7 through B0, B7 being the high order bit. Some module and/or printer manufacturers use B8 through B1, others use either of those to represent the bits in reverse order, B0 or B1 being the highest order bit, and B7 or B8 the lowest! To compound things further, the microprocessor manufacturers can't even agree on how to store a "double byte" in memory. Motorola uses Most significant - Least significant with the highest order byte stored at the lowest address. Some of the others use the reverse order, the low order byte being stored at the lowest address, the equivalent of storing the hexadecimal number 1234 so that when read from left to right it reads 3412! This results in some confusion in storing ASCII text in memory. A 16 bit word stores two characters, so the word "FLEX" for example, is stored in memory in order from lower to higher memory address as "LFXE". The whole thing looks like a conspiracy so that those who don't know will require a great deal of help from those who "do know." Actually, all that happened is that computers came along and breakthroughs in price/performance ratio happened so fast

that each company simply developed a standard of its own without giving any thought to the possibility that someone else would do it differently. The situation is simply a result of the very rapid development of the technology.

Perhaps that is an oversimplification. Good old Big Blue seems to go out of its way to avoid doing things in a standard way. Long after most of the computer and peripheral manufacturers had standardized on ASCII codes, IBM still insisted on its EBCDIC code. ASCII is an acronym for American Standard Code for Information Interchange. The name EBCDIC is an acronym for something that I've seen, but don't have at hand, probably something like Extended Binary Code for Digital Information Communication or the like. At any rate, I know of a company that owes its existence to this. The company builds interface boxes to allow IBM owners to use standard peripherals. At the simplest, they build two-way ASCII/EBCDIC code converters. Actually, their boxes do more than that, in that they are also protocol converters.

Years ago, Digital Equipment Company built their PDP-8 series of computers. They were 12 bit machines, and DEC used Octal notation rather than Hexadecimal. Octal was fine for 12 bit machines because 12 is divisible by 3. However, when 16 bit machines came along, people started thinking of 16 bits as a "word" made up of two 8 bit portions called "bytes." It is much easier to think of these in terms of groups of four bits as in Hexadecimal notation. Take for example, the following:

```
0100 0011 0010 0001      Hexadecimal 4321
```

Now split it into 3 bit chunks for Octal

```
0 100 001 100 100 001      Octal      041441
```

Let's do it again, representing the 16 bit word as two 8 bit bytes:

```
0100 0011      0010 0001      Hexadecimal 43  21
```

```
01 000 011      00 100 001      Octal      103 041
```

As you can see, working with Octal codes, 16 bit words, and 8 bit bytes is thoroughly confusing! Hexadecimal representations nicely split in the middle of the word to make two bytes.

### **PAT Bug**

My friend Albert McDaniel at the University of Maine called the other night. He said that he had found a way to cause PAT to lock up the computer. All presently distributed copies of PAT have this problem. If you have the PL/9 6809 version and you own PL/9 you can fix it easily. The same goes for the 68000 version in "C" if you have the "C" compiler.

The symptom is just this: If you backspace from a line to the previous one and then hit ESC ^E to delete to the end of the line, and if there are no characters to the right of the cursor, the computer will LOCK UP.

The cure is to find the procedure ESCAPES. In the PL/9 version it is in the COMMANDS.LIB file. The "C" version is all in one file. In either case just find the procedure (I should say the function in "C"). Now find: CASE 5. The PL/9 version is:

```
WHILE CH <> CR BEGIN
```

Change this to:

```
WHILE SCREEN(CURCHR) <> CR .AND SCREEN(CURCHR) <> 0 BEGIN
```

The "C" version:

```
while(screen[curchr] != cr)
```

Should be changed to:

```
while(screen[curchr] != cr && screen[curchr] != 0)
```

The problem was that at the end of a line of text, what follows is a large number of nulls. Just exactly why the result was not simply to delete the next whole line rather than locking up the computer, I am not sure, but the change causes nothing at all to happen if there are no more characters to the right of the cursor. The cursor simply returns to where it was. If you don't have the appropriate compiler to make the change, just be warned that there is a problem. I suspect that this has gone undetected for so long because it is not something you would ordinarily do while editing. It doesn't make sense to delete to the end of the line when you are already AT the end of the line. Nevertheless, good software should be tolerant of key sequences that don't make sense if only to guard against peculiar things happening when a key is struck by accident.

### **Self Analysis Time**

I recently wrote a letter to Don Williams indicating that I would no longer be a regular contributor to '68' Micro Journal. No, I'm not mad at anyone for anything. It's just that not much is happening to write about, either in the way of new developments or in my computer applications. I see clearly that the 6809 is reaching the end of its life as an interesting subject, at least to me. After using the newer, bigger, faster systems, I find it hard to be satisfied with the 6809 system anymore. At work, our largest program used to be 4K or 6K and wonderful PL/9 would compile such a program on our 2 MHz system with 8" floppy disk drives in a minute or so, quite tolerable and considerably faster than most of the other compilers available then and now.

Lately we have been going on to bigger and better programs with keyboard operator interface, lots of menus and prompts etc. Our programs have grown to 20K on the average, and we are pushing 32K with some of them. Compile time really gets stretched out intolerably. When I compile the same programs on a 68008 or 68020 system using PLuS or "C," the compile times are back down in the one minute range. The company can't afford to have its programmers kill ten minutes or so for each little change in a program, so we must move on to the newer processors. Oh, sure, we could push the speed up by substituting a Hitachi 6309 3 MHz version and putting a hard disk on the system. We could write our own software to tickle a DAT so we could expand memory, but as things stand, we can't edit a text file larger than about 25K in one piece. PAT's source code is over 65K, and it is split into a number of library files in the 6809 version. The Plus and C versions for the 68XXX can all edit themselves in an edit buffer of 100K or so, and the buffer size can be increased to whatever memory can stand. (Of course PAT and several others of the 6809 FLEX editors have the feature of being able to load part of a disk file, edit that, save that portion and load another part of a file from the disk, but if you have ever tried that you will agree it is a pain).

Presently, I am in limbo, that is, the old computers are uninteresting and destined for the museum shelf. The new ones have not yet been applied to our development problems. I don't want to write about the old ones anymore, and I don't know enough about the new ones to write

intelligently about them (yet). Perhaps the switch to the newer ones will set me off again in a flurry of activity to learn how to become an efficient assembler programmer, thoroughly understand other operating systems than FLEX, etc. I presently use OS9 and MS-DOS nearly every day, but I don't understand either to the depth that I understand FLEX and perhaps SK\*DOS.

What I am getting at is that at least for a while, I will be a now-and-then contributor to '68' Micro Journal. I just received the newest version of PLuS on a 3.5 inch disk, so a review of that will have to wait until I can get a copy on a disk that I can read with one of the computers that are available to me. When I can, I will do a complete review on that, and on OmegaSoft Pascal as well. I've given you bits and pieces of reviews of both over the past year or so, and it really isn't fair to the suppliers for me not to give you a complete review. I'll try to approach both as a first time user, though I have used the 6809 versions of them for a long time.

Part of my frustration with writing this column has always been that I am a "comparer". I tend to relate something new to something that I already know about rather than to look at it on its own merits. A good review really ought to discuss software on its own. Does it function as advertised? Does it do some things particularly well? Does it have some weak points? When I was a beginner, I think I was much more objective than now. Presently I have some rather hard and fast opinions with regard to software and how it should work. My tolerance level for something different, that doesn't quite fit my mold or pattern is getting smaller. Perhaps I have become too much a critic. When I first purchased The Technical Systems Consultants Editor (called EDIT) for the 6800 system (on a cassette tape, no less), I marveled at it, since I had never used a text editor before. Anything that would allow me to edit text would have been a wonder at the time. Now I've used a dozen or so editors, and ended up writing my own. I couldn't possibly review an editor objectively anymore!

What can be said about a "C" compiler? If it is reasonably complete, it compiles standard "C" source lines and has a reasonable "Standard Library" it is OK. Of course one can look at compile times, object code size and benchmark execution tests, but that gets pretty old after a while. Is my predicament becoming clearer?

I think I can sum it all up briefly by saying that at least for the present, this stuff is getting to be old to me. Oh, don't think I've curled up and died by any means. I spent a day of the Thanksgiving holiday and a couple of evenings back again at the McCosh "C" compiler for the 6809. I've re-acquainted myself with Whimsical from John Spray, and I have been doing a special version of JUST for the Centronics 737 printer since I managed some time ago to pick up three or four of them in pretty good shape. The 737 was also made as the Atari 825 for use with the older Atari computers. It has a nice proportional spaced type font. A few years ago, Lane Lester reworked JUST in the PL9 version to work with a proportional type font on one of his printers, and I later modified it (primarily changing the character width table) to work with the Centronics. I've recently adapted that version (I call it CJUST for Centronics) to run on the 68000 system by translating it to "C." I've been debugging it more thoroughly thanks again to Albert McDaniel who recently bought a Mustang-08A. Albert has a 737 and he found some bugs in CJUST almost immediately.

At any rate, yes, I am still computing on the 6809 running FLEX, the 68XXX systems running OS9, and the Tandy IBM compatibles running MS-DOS. Lately, though I have had more fun computing than writing about it, since my projects are the kind that draw quite valid criticism from readers as being too involved and/or too specialized.

By the way, a serious bug has been called to my attention in the scientific functions package that I translated from PL9, and that is included in hard copy form in the manual for Windrush "C." The error is trivial to fix, but it causes great problems with the exp() function. Just inside the "start" bracket { for the exp function there is a variable declaration "int k;". Using an integer



for k seriously limits the range over which the exp function works properly. It does not work at all for negative arguments. The declaration should be "double k:". With this small change, the function will work vastly better. I guess this proves that not very many users of 6809 systems actually do any scientific calculations. This library has appeared in the manual for at least 2 1/2 years, and no one has called the bug to my attention before last month when Charles Rook of Phoenix wrote me a letter describing the problem (and the cure). Thanks, Charles!

As I get back at this I note that nearly a month has gone by since I wrote the last paragraph above. I expect that I will continue to be a contributor to '68' on a less regular basis than previously, as long as this publication exists. I've made some progress toward having a 68008 system running very reliably and I am about to launch into the world of 68XXX assembler programming, the learning of which is an absolute necessity before the company switches to those processors. I would feel it to be a terrible handicap to have to program only in a higher level language without any knowledge of assembler programming techniques. Of course switching from the 6809 to the 68XXX is not like starting from scratch.

Since I've already found it possible to write little utility programs for the 68XXX in "C" or PLuS, I'll probably be doing some of that along with learning Assembler. Excuse me, Don but I will also be going in another direction just a little (for my own satisfaction primarily). I have found that the newest update of Turbo Pascal for the IBM compatibles has enough of an interface with the screen to make it possible to translate PAT into it. There are Pascal procedures to clear the screen, place the cursor at a screen position, erase to the end of a line, etc. I have one reservation (or maybe two) about that as a project. First, I expect the executable file will be quite large, and secondly, I have some doubts about the execution speed. Maybe I'll be pleasantly surprised. At any rate, one of the projects on my list is to translate enough PAT to Turbo Pascal to see if the result would be usable. I'd really like to get all the systems that I use running the same editor. That sure would make life simpler!

As I write this, I have been on an impossible work project for the last two months. Time there has kept me from much computing that was not directly related to the project (both at work and at home). The one exception has been a little diversion I started to keep my sanity. I've been translating the MONOPOLY program from "101 Games in BASIC" to PL9 and improving it as I go. I am hoping we are about at the end of the big work project so I can start to think about other things again and get back to some "exploratory" computing with the 68XXX systems again. Should that happen, or perhaps I should say when that happens, I will most likely be inspired to write more columns.

**Editor's Note:** Ron Anderson has been a regular contributor to 68 Micro Journal since it's first year. We have all come to respect his expertise and appraisals, concerning both software and hardware. His column was the longest continuous running computer magazine column I am aware of. It can be truthfully said - "Ron Anderson is a true pioneer in the micro computer world." And we will all miss his monthly visits.

So I believe I not only speak for myself and the entire staff here at Computer Publishing, Inc., but for all of our readers when I say - **"Don't stay away too long Ron, we all thank you for your sharing and look forward to your next visit!"**

DMW

FOR THOSE WHO NEED TO KNOW

68 MICRO  
JOURNAL™

# Logically Speaking

Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

## The Mathematical Design of Digital Control Circuits

By: R. Jones  
Micronics Research Corp.  
33383 Lynn Ave., Abbotsford, B.C.  
Canada V2S 1E2  
Copyrighted © by R. Jones & CPI

### Mile 8, heading towards Mile 9

#### ELEMENTARY MERGING

Looking at Diagram 26 of our previous lesson, we're going to see if we can reduce the number of rows in the flow-table by merging (or combining) specific rows. The key lies in the Box-As of each row. Our first step is to check for EQUIVALENCES by commencing with row 1, noting its Box-A pattern (namely 1,2,8) and scanning the patterns for all succeeding rows for an identical pattern. Out of luck! So we note row 2's pattern (3,2,-) and search for a match in all succeeding rows. Again no luck! So we try checking row 3, row 4, etc. all the way through the table. Not a match to be found anywhere!! But we don't despair! Having found no direct equivalences, we'll look for PSEUDO-EQUIVALENCES, or false equivalences, instead.

How do we do that? you ask. Well, we've already observed that in our flow-table there are several empty addresses INTO WHICH THE CIRCUIT ACTION DOES NOT ENTER. As these are (impossible) phi-states, we may make them anything we choose without affecting the circuit action. Let's consider rows 2 and 10 as an example of what we have in mind! Their row-patterns read 3,2,- and 3,-,10 respectively (where '-' signifies phi), so we'll make them both read 3,2,10 by changing row 2's '-' into a 10 and row 10's '-' into a 2. One further step! Imagine being able to lift up row 10 completely and superimpose it on row 2. IF, AND ONLY IF, THE BOX-Cs ARE COMPATIBLE AS WELL, WE'VE GOT A PSEUDO-EQUIVALENCE. That is, we MUST NOT find a '1' overlaid on a '0', or vice-versa. It's quite OK for a 1 or a 0 (or even phi) to overlay a phi, as these are not incompatible, but a 1 or a 0 must then take precedence over a phi.

There's no conflict at all between row 2 and row 10, so we'll decide that these two rows are in fact pseudo-equivalent, ie row 10 is equivalent to row 2. Row 10's first two Box-Cs are subordinate to those of row 2, but address 01.10 gets inserted into address 01.2, and row 10 itself eliminated. Finally, as row 10 is the same as row 2, we go through the flow table and change all 10s occurring in Box-A to 2s. If you now check out the action commencing at address 00.5, you'll find that you'll correctly end up at address 00.3, JUST AS BEFORE.

So now we're down to 10 rows only! Further checks indicate that rows 4 and 11 can also be merged in this way, by forcing them into the pattern 5,4,11, as can rows 6 and 8 by forcing them into the pattern 7,6,8.

We've only got 8 rows now, with no further merging possible, but 8 rows means we only need 3 relays to do the job instead of the original 4. Merging, by the way, need not be restricted to only 2 rows at a time. ANY NUMBER of rows which are equivalent or pseudo-equivalent may be merged! For example, three rows with the patterns 1,2,3 and 1,-,3 and 1,2,- may be merged into one row with the pattern 1,2,3. We'll deal more fully with this a little later when we'll consider merging in much more detail.

At this stage, we've got an 8-row flow-table with rows numbered 1, 2, 3, 4, 5, 6, 7 and 9, so for the sake of neatness we finally change all black-9s to black-8, ending up with the flow-table of Diagram 27a (ignoring for the moment the "red" numbers in Box-Bs).

$X_1 X_2$		00	01	10	11	16	17	18	19
1	0	1	0	2	16	6	8		
	0	000	000	000	000				
2	2	3	2	2	18	2	10		
	2	000	000	000	010				
3	3	3	6	4	22	8	14		
	6	100	100	100					
4	4	5	7	4	23	4	15		
	7	000	100	001					
5	5	5	3	6	19	2	11		
	3	010	010	010					
6	6	7	1	6	17	6	9		
	1	000	010	000					
7	7	7	5	7	21	4	13		
	5	001	001	001					
8	8	1	4		20	8	12		
	4	000		100					

(a)

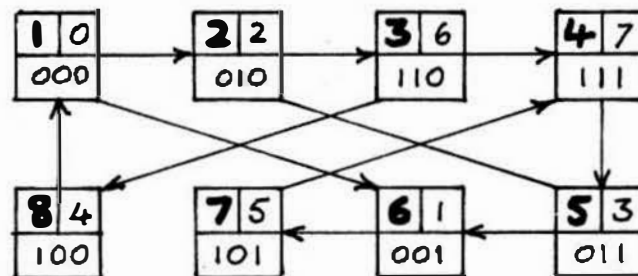


Diagram 27

(b)

### CONSTRUCTING THE STATE-DIAGRAM

Holding to our normal procedure, the next step is to draw up a state-diagram of eight units, whose Box-As are numbered from 1 to 8 in black, as shown in Diagram 27b, and then to add the arrows in this manner :

Commencing with row 1 of the FLOW-TABLE, we note that there are two ways out of this row (keep an eye on the Box-As), one of which is to row 2 and the other to row 6, so in the state-diagram we draw an arrow pointing FROM unit 1 TO both units 2 and 6. Row 2 has only one way out, to row 3, so we draw an arrow FROM unit 2 TO unit 3 in the state-diagram. Row 3 goes to rows 4 and 8, and so on until we have the rather complicated pattern of arrows of Diagram 27b.

### GRAY-CODING THE STATE-DIAGRAM

Knowing that we have 3 relays to play with, we must now Gray-code all the Box-Cs of the state-diagram, which means that we have to insert a 3-bit binary code (one bit for each relay) such that no two units connected by an arrow will differ by more than one bit-position. A first attempt would be to code the loop 1-2-3-8-1 with the familiar 00, 01, 11, 10 pattern for the first 2 bit-positions and a permanent 0 for bit-3. Then we'd try the loop 2-3-4-5-2 (mainly because two of them are already coded), noting that in proceeding from 1-2-3 we've gone through the pattern "no relays energised", "one relay energised" and "two relays energised". So let's preserve this pattern (though we're not BOUND to do so) by entering 111 in unit 4, which leaves us no choice but to insert 011 in unit 5, as unit 5 MUST be one-bit different from both units 4 and 2, and 110 has already been used up in unit 3.

That's 6 down, 2 to go! Let's tackle the loop 4-5-6-7-4, keeping an eye on the fact that unit 6 must not only be one bit different from unit 5 but also from unit 1, which leaves us little alternative but to insert 001. This in turn compels us to enter 101 in unit 7 to keep it one bit different from both units 4 and 6.

Our state-diagram is now completely Gray-coded, and we can therefore insert the decimal equivalents of these codings (in red) in Section B of all units, and then transfer these figures to the outside left-hand column of the flow-table itself.

# SCULPTOR

From the world's oldest & largest OS-9 software house!



**CUTS PROGRAMMING TIME UP TO 80%**  
**6809/68000-68030 Save 70%**

SCULPTOR-4GL S.E.MEDIA VERSION, A "SPECIAL" S.E. Media Version of OS-9 SCULPTOR. OS-9 levels one and two (three GIMIX) 6809, all 68XXX OS-9 standard systems. Equivalent to regular SCULPTOR versions 1.4:8. One of if not the most efficient and easy to develop DBMS type systems running under OS-9! A system of flexible keyed file access that allows extremely fast record and data retrieval, insertion and deletion or other programmed modifications. Access by key or in ascending order, very fast. The system provides automatic menu generation, compilation and report generation. Practically unlimited custom input format and report formatting. A rich set of maintenance and repair utilities. An extremely efficient development environment that cuts most programming approximately 80% in development and debugging! Portable, at source level, to MS-DOS, UNIX and many other languages and systems.

Standard Version: 1.6 6809 - \$1295.00  
68000 \$1295.00  
68020 \$1990.00

Due to a "Special One Time" Purchase, We  
Are Making This Savings Offer. Quantities  
Limited!

*Once this supply is gone - the price goes  
back up!*

System OS-9: 6809/68000-68030

• Regular ~~\$1295.00~~

ONLY

**\$295.00**

+ \$7.50 S&H USA  
Overseas - Shipped Air Mail  
Collect

**S.E. MEDIA**

POB 849

5900 CASSANDRA SMITH ROAD  
HIXSON, TN 37343 615 842-4601



**AVE - WHILE SUPPLIES LAST!**



Telephone: (615) 842-4600

# South East Media

## OS-9, UniFLEX, FLEX, SK\*DOS

### SOFTWARE

Telex: 5106006630

!!! Please Specify Your Operating System and Disk Size !!!

# SCULPTOR

Full OEM & Dealer Discounts Available!

## THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

## AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

## SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS-DOS, Unix, Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

## APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and - without any alterations to the programs - run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

## SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

## INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australia, the Americas and Europe - Sculptor is already at work in over 20 countries.

## THE PACKAGE

With every development system you receive:

- ☐ A manual that makes sense
- ☐ A periodic newsletter
- ☐ Screen form language
- ☐ Report generator
- ☐ Menu system
- ☐ Query facility
- ☐ Set of utility programs
- ☐ Sample programs

For resale products, the run-time system is available at a nominal cost.

**Facts**  
■■■■■

**Features**  
■■■■■■■

## DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:

- ☐ alphanumeric
- ☐ integer
- ☐ floating point
- ☐ money
- ☐ date

## DATA FILE STRUCTURE

- ☐ Packed, fixed-length records
- ☐ Money stored in lower currency unit
- ☐ Dates stored as integer day numbers

## INDEXING TECHNIQUE

Sculptor maintains a B-tree index for each data file. Program logic allows any number of alternative indexes to be coded into one other file.

## INPUT DATA VALIDATION

Input data may be validated at three levels:

- ☐ automatic by field type
- ☐ validation list in data dictionary
- ☐ programmer coded logic

## ARITHMETIC OPERATORS

- Unary minus
- \* Multiplication
- / Division
- % Remainder
- + Addition
- Subtraction

## MAXIMA AND MINIMA

- Minimum key length 1 byte
- Maximum key length 160 bytes
- Minimum record length 3 bytes
- Maximum record length 32767 bytes
- Maximum fields per record 32767
- Maximum records per file 16 million
- Maximum files per program 16
- Maximum open files

Operating system limit

## PROGRAMS

- ☐ Define record layout
- ☐ Create new indexed file
- ☐ Generate standard screen form program
- ☐ Generate standard report program
- ☐ Compile screen-form program
- ☐ Compile report program
- ☐ Screen-form program interpreter
- ☐ Report program interpreter
- ☐ Menu interpreter

## RELATIONAL OPERATORS

- = Equal to
- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to
- <> Not equal to
- and Logical and
- or Logical or
- contains Contains
- begins with Begins with

## SPECIAL FEATURES

- ☐ Full date arithmetic
- ☐ Echo suppression for passwords
- ☐ Terminal and printer independence
- ☐ Parameter passing to sub-programs
- ☐ User definable date format

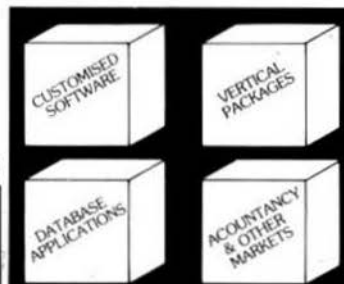
## QUERY FACILITY

- ☐ Query facility
- ☐ Reformat file
- ☐ Check file integrity
- ☐ Backup/restore
- ☐ Alter language and data format
- ☐ Setup terminal characteristics
- ☐ Setup printer characteristics

## SCREEN FORM LANGUAGE

- ☐ Programmer defined options and logic
- ☐ Multiple files open in one program
- ☐ Default or programmer processing of exception conditions
- ☐ Powerful verbs for input, display and file access
- ☐ Simultaneous display of multiple records
- ☐ Facility to call sub-programs and operating system commands
- ☐ Conditional statements
- ☐ Subroutines
- ☐ Independent of arrival type

**Sculptor for 68020  
OS-9 & UniFLEX  
\$995**



MUSTANG-020 Users - Ask For Your Special Discount!

**MUSTANG-020 \*\$1,990 \$398 \$795 PC/XT/AT/MSDOS \$695 \$139 \$299**

**MUSTANG-08 \*\$1,295 \$259 \$495 Call or write for prices on the following systems.**

XENIX SYS III & V, MS-NET, UNIX SYS III & V, ATARI OS-9, 68K, UNOS, ULITUX/VMS (VAX, REGAL, STRIDE, ALTOS, A'RICORT, ARETE, ARM-STRONG, BLUEDALE, CHARLES RIVERS, GMX, CONVERG, TECH, DEC, CIPHER, EQUINOX, GOULD, III, HONEYWELL, IBM, INTEL, MEGADATA, MOTOROLA, NCR, NIXDORF, N-STAR, OLIVETTI/AT&T, ICL, PERKINS ELMER, PHILIPS, PIXEL, PLESSEY, PLEXUS, POSITRON, PRIME, SEQUENT, SIEMENS, SWTPC, SYNNEX, TANDY, TORCH, UNISYS, ZYLOG, ETC.

**\* For SPECIAL LOW SCULPTOR prices especially for 68096RXXX OS-9 Systems - See Special Ad this issue. Remember, "When they are gone the price goes back up as above!"**

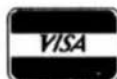
... Sculptor Will Run On Over 100 Other Types of Machines ...

... Call for Pricing ...

!!! Please Specify Your Make of Computer and Operating System !!!

- \* Full Development Package
- \*\* Run Time Only
- \*\*\* C Ke File Library

**Availability Legend**  
O = OS-9, S = SK\*DOS  
F = FLEX, U = UniFLEX  
CC = Color Computer OS-9  
CCF = Color Computer FLEX



**South East Media**  
5900 Cassandra Smith Rd. - Hixson, TN, 37343  
Telephone: (615) 842-4600 Telex: 5106006630



**\*\* Shipping \*\***  
Add 2% U.S.A. (min. \$3.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola. \*FLEX and UniFLEX are Trademarks of Technical Systems Consultants. \*SK\*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK\*DOS

## ASSEMBLERS

**ASTRUK09** from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, S, CCF - \$99.95

**Macro Assembler for TSC** -- The FLEX, SK\*DOS STANDARD Assembler.

Special -- CCF \$35.00; F, S \$50.00

**OSM Extended 6809 Macro Assembler** from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX, SK\*DOS.

FLEX, SK\*DOS, CCF, OS-9 \$99.00

**Relocating Assembler/Linking Loader** from TSC. -- Use with many of the C and Pascal Compilers.

F, S, CCF \$150.00

**MACE**, by Graham Trout from Windvush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.

F, S, CCF - \$75.00

**XMACE** -- MACE w/Cross Assembler for 6800/1/2/3/8

F, S, CCF - \$98.00

## DISASSEMBLERS

**SUPER SLEUTH** from Computer Systems Consultants Interactive Disassembler. extremely **POWERFUL!** Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)

CCD (32K Req'd) Obj. Only \$49.00

F, S, \$99.00 - CCF, Obj. Only \$50.00 U, \$100.00

CCF, w/Source \$99.00 O, \$101.00

CCO, Obj. Only \$50.00

OS9 68K Obj. \$100.00 w/Source \$200.00

**DYNAMITE+** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only \$100.00 - CCO, Obj. \$ 59.95

F, S, " " \$100.00 - O, object only \$150.00

U, " " \$300.00

## CROSS ASSEMBLERS

**TRUE CROSS ASSEMBLERS** from Computer Systems Consultants --

Supports 1802/5, Z-80, 6800/1/2/3/8/11/HCI1, 6804, 6805/HCI05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/3/5/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems.

Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.

68000 or 6809, FLEX, SK\*DOS, CCF, OS-9, UniFLEX

any object or source each - \$50.00

any 3 object or source each - \$100.00

Set of ALL object \$200.00 - w/Source \$500.00

**XASM Cross Assemblers** for FLEX, SK\*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.

Complete set, FLEX, SK\*DOS only - \$150.00

**CRASMB** from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK\*DOS (binary). Written in Assembler ... e.g. **Very Fast.**

**CPU TYPE - Price each:**

For:	MOTOROLA	INTEL	OTHER COMPLETE SET
FLEX9	\$150	\$150	\$399
SK*DOS	\$150	\$150	\$399
OS9/6809	\$150	\$150	\$399
OS9/68K	-----	-----	\$432

**CRASMB 16.32** from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.

FLEX, SK\*DOS, CCF, OS-9/6809 \$249.00

## COMMUNICATIONS

**C-MODEM Telecommunications Program** from Computer Systems

Consultants, Inc. -- Menu-Driven: supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem?" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

FLEX, SK\*DOS, CCF, OS-9, UniFLEX, 68000 & 6809 with Source \$100.00 - without Source \$50.00

**X-TALK** from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.

X-TALK Complete (cable, 2 disks) \$99.95

X-TALK Software (2 disks only) \$69.95

X-TALK with C-MODEM Source \$149.95

**XDATA** from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.

U - \$299.99

Availability Legend  
O = OS-9, S = SK\*DOS  
F = FLEX, U = UniFLEX  
CCO = Color Computer OS-9  
CCF = Color Computer FLEX



South East Media

5900 Cassandra Smith Rd. - Hixson, In. 37343



\*\* Shipping \*\*  
Add 2% U.S.A. (min. \$2.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola.\*FLEX and UniFLEX are Trademarks of Technical Systems Consultants.\*SK\*DOS is a Trademark of Star-K Software Systems Corp.



Telephone: (615) 842-4600

**South East Media**  
OS-9, UniFLEX, FLEX, SK-DOS

Telex: 5106006630

## PROGRAMMING LANGUAGES

**PL/9** from Windrush Micro Systems -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. \$500+ page Manual with tutorial guide.

F, S, CCF - \$198.00

**PASC** from S.E. Media - A FLEX9, SK-DOS Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package come complete with source (written in PASC) and documentation.

FLEX, SK-DOS \$95.00

**WHIMSICAL** from S.E. MEDIA Now supports Real Numbers. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9.

F, S and CCF - \$195.00

**KANSAS CITY BASIC** from S.E. Media - Basic for Color Computer OS-9 with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFTS, RIGHTS, MIDS, STRINGS, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 \$39.95

**C Compiler** from Windrush Micro Systems by James McCosh. Full C for FLEX, SK-DOS except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries.

F, S and CCF - \$295.00

**C Compiler** from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.

FLEX, SK-DOS, CCF, OS-9 (Level II ONLY), U - \$575.00

**PASCAL Compiler** from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F, S and CCF 5" - \$190.00 F, S 8" - \$205.00

**PASCAL Compiler** from OmegaSoft (now Certified Software) -- For the PROFESSIONAL; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. OS-9, F, S and CCF - \$550.00 OS-9 68000 Version - \$900.00

**KIBASIC** - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

FLEX, SK-DOS, CCF, OS-9 Compiler /Assembler \$99.00

**CRUNCH COBOL** from S.E. MEDIA -- Supports large subset of ANSI Level 1 COBOL with many of the useful Level 2 features. Full FLEX, SK-DOS File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. A very popular product.

FLEX, SK-DOS, CCF - \$99.95

**FORTH** from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - \$58.95

**FORTHBUILDER** is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs. All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.

FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change. Like compilers for other languages, FORTHBUILDER can operate in "batch mode".

The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words. FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.

F, CCF, S - \$99.95

## EDITORS & WORD PROCESSING

**JUST** from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.COM supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Grafix); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

\* Now supplied as a two disk set:

Disk #1: JUST2.COM object file.

JUST2.TXT PL/9 source: FLEX, SK-DOS - CC

Disk #2: JUST2SC object and source in C:

FLEX, SK-DOS - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C

**Availability Legend**  
O = OS-9, S = SK-DOS  
F = FLEX, U = UniFLEX  
CC = Color Computer OS-9  
CCF = Color Computer FLEX



**South East Media**  
5900 Cassandra Smith Rd. - Hickory, TN. 37343



**\*\* Shipping \*\***  
Add 2% U.S.A. (min. \$2.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola. \*FLEX and UniFLEX are Trademarks of Technical Systems Consultants. \*SK-DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

## South East Media

OS-9, UniFLEX, FLEX, SK\*DOS

Telex: 5106006630

source compiles to a standard syntax JUST.COMD object file. Using JUST syntax (.p, .u, .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL-9 FLEX only: F, S & CCF - \$49.95

Disk Set (2) - F, S & CCF & OS9 (C version) - \$69.95

OS-9 68K000 complete with Source - \$79.95

**PAT** from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX, SK\*DOS \$129.50

\* SPECIAL INTRODUCTION OFFER \* \$79.95

SPECIAL PAT/JUST COMBO (w/source)

FLEX, SK\*DOS \$99.95

OS-9 68K Version \$229.00

SPECIAL PAT/JUST COMBO 68K \$249.00

Note: JUST in "C" source available for OS-9

**CEDRIC** from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassle' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - approx. 14,000 plus of free memory! Extra fun for programming as well as text.

FLEX, SK\*DOS \$69.95

**BAS-EDIT** from S.E. Media - A TSC BASIC or X BASIC screen editor. Appended to BASIC or X BASIC, BAS-EDIT is transparent to normal BASIC/X BASIC operation. Allows editing while in BASIC/X BASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/X BASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

FLEX, CCF, SK\*DOS \$39.95

**SCREDDITOR III** from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX, SK\*DOS or SSB DOS, OS-9 - \$175.00

**SPELLB** "Computer Dictionary" from S.E. Media -- OVER 150,000 words! Look up a word from within your Editor or Word Processor (with the SPELL.COMD Utility which operates in the FLEX, SK\*DOS UCS). Or check and update the Text after entry: ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F, S and CCF - \$129.95

**STYLO-GRAPH** from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK\*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - \$99.95,

F, S or O - \$179.95, U - \$299.95

**STYLO-SPELL** from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - \$69.95,

F, S or O - \$99.95, U - \$149.95

**STYLO-MERGE** from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES 6809 CCF and CCO - \$59.95,

F, S or O - \$79.95, U - \$129.95

**STYLO-PAK** --- Graph + Spell + Merge Package Deal!!!

F, S or O - \$329.95, U - \$549.95

O, 68000 \$695.00

## DATABASE ACCOUNTING

**XDMS** from Westchester Applied Business Systems  
FOR 6809 FLEX-SK\*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed names! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

**XDMS-IV** Data Management System

**XDMS-IV** is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

**POWERFUL COMMANDS!**

**XDMS-IV** combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriented which allows almost instant implementation of a process design.

**SESSION ORIENTED!**

**XDMS-IV** is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV.

**Availability Legend**  
O = OS-9, S = SK\*DOS  
F = FLEX, U = UniFLEX  
CCO = Color Computer OS-9  
CCF = Color Computer FLEX



South East Media  
5900 Cassandra Smith Rd. - Hixson, TN. 37343



**\*\* Shipping \*\***  
Add 2% U.S.A. (min. \$2.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola.\*FLEX and UniFLEX are Trademarks of Technical Systems Consultants.\*SK\*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK-DOS

#### IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database file and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK-DOS(5/8") \$249.95

### UTILITIES

**Basic09 XREF** from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.

O & CCO obj. only -- \$39.95; w/ Source - \$79.95

**BTee Routines** - Complete set of routines to allow simple implementation of keyed files - for your programs - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.

O & CCO obj. only - \$89.95

**Lucidata PASCAL UTILITIES** (Requires Pascal ver 3)

**XREF** -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

**INCLUDE** -- Include other Files in a Source Text, including Binary - unlimited nesting.

**PROFILER** -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, S, CCF --- EACH 5" - \$40.00, 8" - \$50.00

**DUB** from S.E. Media -- A UniFLEX BASIC decompiler Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.

U - \$219.95

**LOW COST PROGRAM KITS** from Southeast Media The following kits are available for FLEX, SK-DOS on either 5" or 8" Disk.

#### 1. BASIC TOOL-CHEST \$29.95

**BLISTER.CMD**: pretty printer  
**LINXREF.BAS**: line cross-referencer  
**REMPAC.BAS**, **SPCPAC.BAS**, **COMPAC.BAS**:  
remove superfluous code

**STRIP.BAS**: superfluous line-numbers stripper

#### 2. FLEX, SK-DOS UTILITIES KIT \$39.99

**CATS**. CMD: alphabetically-sorted directory listing  
**CATD.CMD**: date-sorted directory listing  
**COPYSORT.CMD**: file copy, alphabetically  
**COPYDATE.CMD**: file copy, by date-order  
**FILEDATE.CMD**: change file creation date  
**INFO.CMD** (& **INFOGMOX.CMD**): tells disk attributes & contents  
**RELINK.CMD** (& **RELINK82**): re-orders fragmented free chain  
**RESQ.CMD**: undeletes (recovers) a deleted file  
**SECTORS.CMD**: show sector order in free chain  
**XL.CMD**: super text lister

#### 3. ASSEMBLERS/DISASSEMBLERS UTILITIES \$39.95

**LINEFEED.CMD**: 'modularise' disassembler output  
**MATH.CMD**: decimal, hex, binary, octal conversions  
& tables

**SKIP.CMD**: column stripper

#### 4. WORD - PROCESSOR SUPPORT UTILITIES \$49.95

**FULLSTOP.CMD**: checks for capitalization

**BSTYCTT.BAS** (.BAC): Stylo to dot-matrix printer

**NECPRINT.CMD**: Stylo to dot-matrix printer filter code

#### 5. UTILITIES FOR INDEXING \$49.95

**MENU.BAS**: selects required program from list below

**INDEX.BAC**: word index

**PHRASES.BAC**: phrase index

**CONTENT.BAC**: table of contents

**INDXSORT.BAC**: fast alphabetic sort routine

**FORMATR.BAC**: produces a 2-column formatted index

**APPEND.BAC**: append any number of files

**CHAR.BIN**: line reader

**BASIC09 TOOLS** consist of 21 subroutines for Basic09.

6 were written in C Language and the remainder in assembly.

All the routines are compiled down to native machine code which makes them fast and compact.

1. **CFILL** -- fills a string with characters
2. **DPEEK** -- Double peek
3. **DPOKE** -- Double poke
4. **FPOS** -- Current file position
5. **FSIZE** -- File size
6. **FTRIM** -- removes leading spaces from a string
7. **GETPR** -- returns the current process ID
8. **GETOPT** -- gets 32 byte option section
9. **GETUSR** -- gets the user ID
10. **GTIME** -- gets the time
11. **INSERT** -- insert a string into another
12. **LOWER** -- converts a string into lowercase
13. **READY** -- Checks for available input
14. **SETPRIOR** -- changes a process priority
15. **SETUSR** -- changes the user ID
16. **SETOPT** -- set 32 byte option packet
17. **STIME** -- sets the time
18. **SPACE** -- adds spaces to a string
19. **SWAP** -- swaps any two variables
20. **SYSCALL** -- system call
21. **UPPER** -- converts a string to uppercase

For OS-9 - \$44.95 - Includes Source Code

See Review in January 1987 issue of 68 Micro Journal

### SOFTTOOLS

The following programs are included in object form for immediate application. PL/9 source code available for customization.

**READ-ME** Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.

**CONFIG** one time system configuration.

**CHANGE** changes words, characters, etc. globally to any text type file.

**CLEANTXT** converts text files to standard FLEX, SK-DOS files.

**COMMON** compare two text files and reports differences.

**COMPARE** another check file that reports mis-matched lines.

**CONCAT** similar to FLEX, SK-DOS append but can also list files to screen.

**DOCUMENT** for PL/9 source files. Very useful in examining parameter passing aspects of procedures.

Availability Legend  
O = OS-9, S = SK-DOS  
F = FLEX, U = UniFLEX  
CC9 = Color Computer OS-9  
CCF = Color Computer FLEX



South East Media  
5900 Cassandra Smith Rd. - Hixson, Tn. 37343



\*\* Shipping \*\*  
Add 2% U.S.A. (min. \$2.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola.\*FLEX and UniFLEX are Trademarks of Technical Systems Consultants.\*SK-DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK-DOS

ECHO echos to either screen or file.

FIND an improve find command with "pattern" matching and wildcards.

Very useful.

HEX dumps files in both hex and ASCII.

INCLUDE a file copy program that will accept "includes" of other disk files.

KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.

LISTDIR a directory listing program. Not super, but better than CAT.

MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted.

Very fast. Very useful.

MULTICOL width of page, number of columns may be specified. A MUST!

PAGE similar to LIST but allows for a page header, page width and depth.

Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.

REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!

SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.

SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough. sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.

TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.

TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.

UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.

WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source.

Complete set SPECIAL INTRO PRICE:

5-1/4" w/source FLEX - SK-DOS - \$129.95

w/o source - \$79.95

8" w/source - \$79.95 - w/o source \$49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants

- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F, S and CCF, U - \$25.00, w/ Source - \$50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 \$69.95

## DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

Level I OS-9 obj. \$79.95; w/ Source \$149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes:

REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK-DOS Format so it can be used normally by FLEX, SK-DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK-DOS Directory, Delete FLEX, SK-DOS Files, Copy both directions, etc. FLEX, SK-DOS users use the special disk just like any other FLEX, SK-DOS disk

O - 6809/68000 \$79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.

OS-9 \$85.00

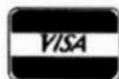
HIER from S.E. Media - HIER is a modern hierarchical storage system for users under FLEX, SK-DOS. It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. Using HIER a regular (any) FLEX, SK-DOS disk (8 - 5" hard disk) can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day solution that all current large systems use. Each directory looks to FLEX, SK-DOS like a regular file, except they have the extension '.DIR'. A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special instal. package is included to install HIER to your particular version of FLEX, SK-DOS. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

FLEX - SK-DOS \$79.95

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX, SK-DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included. ALL 4 Programs (FLEX, SK-DOS, 8" or 5") \$99.50

Availability Legend  
O = OS-9, S = SK-DOS  
F = FLEX, U = UniFLEX  
CC1 = Color Computer OS-9  
CC2 = Color Computer FLEX



South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343



\*\* Shipping \*\*  
Add 2% U.S.A. (min. \$2.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola. \*FLEX and UniFLEX are Trademarks of Technical Systems Consultants. \*SK-DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK-DOS

**COPYCAT** from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SK-DOS, SSB DOS68, and Digital Research CP/M Disks while operating under SK-DOS, FLEX1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F, S and CCF 5" - \$50.00 F, S 8" - \$65.00

**VIRTUAL TERMINAL** from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.

O & CCO - obj. only - \$49.95

**FLEX, SK-DOS DISK UTILITIES** from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX, SK-DOS Utilities for every FLEX, SK-DOS Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten X BASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, X BASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source (either BASIC or A.L. Source Code).

F, S and CCF - \$50.00

BASIC Utilities ONLY for UniFLEX -- \$30.00

### MISCELLANEOUS

**TABULA RASA SPREADSHEET** from Computer Systems Consultants --

TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F, S and CCF, U - \$50.00, w/ Source - \$100.00

**DYNACALC** -- Electronic Spread Sheet for the 6809 and 68000.

F, S, OS-9 and SPECIAL CCF - \$200.00, U - \$395.00

OS-9 68K - \$595.00

**FULL SCREEN INVENTORY/MRP** from Computer Systems Consultants

-- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F, S and CCF, U - \$50.00, w/ Source - \$100.00

**FULL SCREEN MAILING LIST** from Computer Systems Consultants --

The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F, S and CCF, U - \$50.00, w/ Source - \$100.00

**DIET-TRAC Forecaster** from S.E. Media -- An X BASIC program that plans

a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and caloric plan is determined.

F, S - \$59.95, U - \$89.95

### GAMES

**RAPIER** - 6809 Chess Program from S.E. Media -- Requires FLEX, SK-DOS and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels)

F, S and CCF - \$79.95

Availability Legend  
O = OS-9, S = SK-DOS  
F = FLEX, U = UniFLEX  
CC = Color Computer OS-9  
CCP = Color Computer FLEX



South East Media  
5900 Cassandra Smith Rd. - Hixson, TN. 37343



•• Shipping ••  
Add 2% U.S.A. (min. \$2.50)  
Foreign Surface Add 5%  
Foreign Airmail Add 10%  
Or C.O.D. Shipping Only

\*OS-9 is a Trademark of Microware and Motorola. \*FLEX and UniFLEX are Trademarks of Technical Systems Consultants. \*SK-DOS is a Trademark of Star-K Software Systems Corp.

## COMPLETING THE FLOW-TABLE

There are altogether five controls, two primary controls (X1 and X2), and three secondary controls, which we'll call Y1, Y2 and Y3 - that is

$$\begin{matrix} 16 & 8 & 4 & 2 & 1 \\ x1 & x2 & y1 & y2 & y3 \end{matrix}$$

We observe that the binary values 16 and 8 have been forced on X1 and X2, which allows us to head the columns of the flow-table with the appropriate red decimal numbers, after which it becomes a simple matter to complete all Box-Bs of the flow-table (in red) with the sum of each one's coordinates.

## THE PRELIMINARY DECODING-TABLE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	21	22	23
Y <sub>1</sub>		1	1			1	1						1	1	1	1					1	1	1
Y <sub>2</sub>			1	1			1	1			1	1		1		1	1		1			1	1
Y <sub>3</sub>		1		1		1		1	1	1				1		1		1		1	1	1	1
L <sub>1</sub>			∅		∅		1						1		1							1	1
L <sub>2</sub>				1				∅			1	1						1		1			
L <sub>3</sub>		∅				1								1		1					1		

Diagram 28

As before, the next step of the procedure is to construct a table setting out in numerical order the Box-Bs of the the flow-table which contain a red decimal number. Note that minterms 20 and all those higher than 23 do not appear - they are phi-states. Then we tabulate the 1s for the Y energisations, and the 1s and phis for the L outputs.

## DECODING THE FLOW-TABLE

Then begins the task of decoding the decimal minterm numbers in order to arrive at a minimal set of prime implicants for each device. We'll commence with the decoding for Y1.

Y <sub>1</sub>	1	2	5	6	12	13	14	15	21	22	23	16	8	4	2	1	
	x		4									0	0	∅	0	1	∅
	x		4									0	0	∅	1	0	
				x	1		2	1				∅	1	1	∅	∅	28, 30, 29, 31
		16			8			x				∅	∅	1	0	1	29 ✓
			16			8		x				∅	∅	1	1	0	30 ✓
							2	1	x	1	∅	1	∅	∅	∅	∅	31, 29, 20, 30, 28

Diagram 29



You'll notice that this decoding layout is slightly different from previous decoding-tables, in that it has a column to the right headed with a phi. We'll discuss this feature at the appropriate time! The first 2 rows decode quite normally (remember that only 20, and 24 and up are phi's), but when we come to row 3, commencing on minterm-12, and ask "Is  $12 + 16 = 28$  available?" the answer is YES, but it's a phi-minterm. So we make a note in the phi-column to the right that we've made use of 28. "Is  $12 - 8 = 4$  available? No!" Similarly with  $12 - 4 = 8$ , and so we come to  $12 + 2 = 14$  (which IS available) and also  $28 + 2 = 30$  is available, because it's a phi. We therefore put a 2 under minterm-14 and add 30 to our phi-column, NOT FORGETTING TO CHANGE THE 0 IN COLUMN  $y_2$  TO A PHI. If you overlook any part of this procedure, YOUR CIRCUIT IS NOT GOING TO WORK PROPERLY. Similarly for  $12 + 1 = 13$ .

Now, in row 4 of the decoding-table, commencing on minterm-21, we proceed normally (as we did in the previous row) until we come to  $21 + 2 = 23$ , which IS available. We find that we can add 2 to each of the other minterm-numbers, except for minterm-5. Unlike our earlier, simpler procedure, we'll not put a tick yet at the right-hand edge of our decoding-table. First we'll try to transfer our "base-minterm" (that is, our 'x') to another minterm-number. Very often this works out OK! We can transfer provided our new base-minterm fulfills ALL of the following conditions :

1. It MUST be a "blocking" minterm in the row being decoded.
2. The new base must NOT be covered by any previous unticked decoding-row. That is, it must not be covered by an ESSENTIAL prime implicant.
3. It must not "go with" any non-phi variable in the decoding of this row to this point.

Applying condition 1 first, we've only come across one blocking-minterm so far and that is minterm-5, but it fails on the other 2 counts. That is, it's already covered by row 1, and it also "goes with" the non-phi variable  $y_1$ , which produces  $5 - 4 = 1$ , which IS available. That is, seeing as we're blocked in our attempt to change  $y_2$  into a phi, we're forbidden to change a column already tested which "failed" the test, and unfortunately,  $y_1$  has already been tested and failed with our original base.

Under the circumstances, we can't find a new base to which to transfer, so we're now free to put our tick to the right to indicate an ordinary, non-essential prime-implicant, and proceed with the decoding of the remainder of the variable-columns. In the first 3 rows we completed all our runs without coming up against a block, so these three will be ESSENTIAL to our decoding, whereas we may, or may not, need the non-essential rows.

The same kind of situation crops up in row 5, with minterm-6 being ruled out as a new base for reasons similar to those which eliminated minterm-5 in row 4.

Row 6 decodes normally, with no problems, and becomes our fourth essential prime implicant. Examination of the decoding-table shows that the four unticked rows between them cover all the minterms, therefore rows 4 and 5 are redundant and are not incorporated into our Boolean expression for  $Y_1$ , which is

$$\begin{aligned} Y_1 &= X_1' \cdot X_2' \cdot y_2' \cdot y_3 + X_1' \cdot X_2' \cdot y_2 \cdot y_3' + X_2 \cdot y_1 + X_1 \cdot y_1 \\ &= X_1' \cdot X_2' \cdot (y_2' \cdot y_3 + y_2 \cdot y_3') + y_1 \cdot (X_1 + X_2) \end{aligned}$$

#### NOW FOR $Y_2$ 'S DECODING

$Y_2$	2	3	6	7	10	11	13	15	16	18	22	23	$X_1$	$X_2$	$y_1$	$y_2$	$y_3$	$\phi$
	x									16			$\phi$	$\phi$	0	1	0	26
		x	4		8		4						0	$\phi$	$\phi$	1	1	
	4	x							4	16			$\phi$	0	$\phi$	1	0	
✓							x	2					$\phi$	1	1	$\phi$	1	24, 31
✓									x	2	2		1	$\phi$	$\phi$	$\phi$	0	24, 20, 28, 26, 30
														$\phi$	$\phi$	1	1	31
✓	x	x			8	8							0	$\phi$	0	1	$\phi$	
✓		x	1							16	16		$\phi$	0	1	1	$\phi$	

Diagram 30

This one is a little trickier than any we've come across so far, so let's take it gradually. Row 1 commences on minterm-2.  $2 + 16 = 18$  is valid, as is  $2 + 8 = 10$  and  $18 + 8 = 24$ . However, when we come to the  $y_1$  variable, we find that everything "goes" except minterm-10. Normally this would mean placing a tick to the right, BUT if we transfer our base to minterm-10 (which meets ALL the conditions for transfer) there's a possibility that we may be able to convert this prime-implicant into an essential one. We indicate our new base by putting a stroke through the '8' below minterm-10. Proceeding with our decoding,  $10 - 2 = 8$  is not available, but  $10 + 1 = 11$  is. However, we are blocked by minterm-18, and as both minterms 18 and 26 are ruled out as possibilities for a new base we're compelled to put a tick to the right after all. Almost made it, but not quite!! Now for another point to observe - - we'll put a dot over the variable for which we start a run but we're unable to complete, in this case  $y_3$ .

Rows 2, 3 and 6 turn out to be similar to row 1, except that no transfers are possible. Rows 4 and 5 are the only two which prove to be essential prime-implicants. At this stage, with a more primitive method of decoding, we'd consider our decoding to be complete because every minterm is now covered by a decoding row.

Normally, the procedure in larger tables of this sort is to place a small tick or dot above every minterm covered by ESSENTIAL prime-implicants, and then to look for remaining minterms which are covered by only one row. For example, minterm-3 is covered only by row 2, so we'd have to select row 2 as one of our prime-implicants. The selection of this row also covers minterms 7, 11 and 15, which must also have a tick placed over them. Minterm-6 is covered by row 3 alone, so we MUST select this prime-implicant too, which adds minterm-2 to the list of ticked minterms. And so on, until we end up by keeping every prime implicant in the decoding, consisting of two essential prime-implicants and four non-essential ones of three literals apiece.

### AUXILIARY DECODING ROWS

In our refined decoding system, however, we'll not be content with such a large number of non-essentials, so we'll draw a heavy line across the bottom of our table, and try for auxiliary decodings to see if there are better alternatives. First then, we'll tick all minterms covered by ESSENTIAL prime-implicants, and begin again with the lowest unticked minterm, which happens to be minterm-2.

This compares with our original row 1, which also used minterm-2 as its base, but whose satisfactory decoding was blocked by  $y_3$  (dotted). Our technique in this second round of decoding will be to convert  $y_3$  to a phi first, by asking "Is  $2 + 1 = 3$  available?" It is, so we place a 1 under minterm-3, and convert  $y_3$  to a phi. Now we go back to the normal order of decoding and ask "Is  $2 + 16 = 18$  available?" It is, but  $3 + 16 = 19$  isn't, so we transfer to minterm-3 as our base, because minterm-3 is NOT covered by a previous ESSENTIAL prime-implicant. Proceeding with the decoding, we cover minterms 10 and 11, but find ourselves blocked by  $y_1$  when we come to add 4 to each of the minterms already covered in this row. There is no other minterm to which we can transfer, so we place a dot over the blocking variable  $y_1$ , and carry on with variable  $y_2$ . 2 does not go, so this is the end of this row's decoding.

If it were possible to transfer a second time, we would indicate this by changing our original "slash" (marking minterm-3) into an 'X', and "slashing" our newer base-minterm.

We move on to the next lowest unticked minterm, namely minterm-6, and decode in similar fashion. On completion of this row, we find that all minterms are now covered with ONLY TWO NON-ESSENTIAL PRIME-IMPLICANTS OF THREE LITERALS APIECE, instead of the earlier four such prime-implicants.

The final selection of prime-implicants for our decoding is indicated by ticks to the left of the decoding-rows, which gives a Boolean expression for  $Y_2$  of

$$\begin{aligned} Y_2 &= x_2.y_1.y_3 + x_1.y_3' + x_1'.y_1'.y_2 + x_2'.y_1.y_2 \\ &= x_2.y_1.y_3 + x_1.y_3' + y_2(x_1'.y_1' + x_2'.y_1) \end{aligned}$$

We've now covered all aspects of the decoding-system. It's a little more complicated than the simpler, basic system we commenced with, but, as you see, it DOES result in a simpler network, though this will not always be the case, as it sometimes happens that the first decoding was the better after all. But at least we now have a choice in the matter!

### NOW FOR Y3'S DECODING

Y3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	16	8	4	2	1	
	1	3	5	7	8	9	13	15	17	19	21	22	23	X <sub>1</sub>	X <sub>2</sub>	y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>		φ	
	x	4			8	4		16		4				φ	φ	φ	0	1	25, 29	✓	
✓	2	x	2	4				2	16	2	4			φ	0	φ	φ	1			
✓					x	1								φ	1	0	0	φ	24, 25		
✓		2	8				2	x		2	8			φ	φ	1	φ	1	31, 29		
✓											x	1		1	φ	1	1	φ	30, 31		

Diagram 31

Diagram 31 sets out the decoding for Y3, and presents no special problems. Note that in row 1 we transfer to minterm-9 as our base when we find ourselves blocked by y2, only to find ourselves finally blocked by y3. The other four rows decode quite naturally (all of them turning out to be essential) AND ALSO COVER ALL THE MINTERMS, so that row 1 becomes unnecessary after all. Y3's expression is therefore

$$Y3 = X2'.y3 + X2.y1'.y2' + y1.y3 + X1.y1.y2$$

from which we may factor out either y1 or y3 to give a condensed expression.

This disposes of the relay control-functions, leaving only the networks for the lights to decode, so here goes.

### FINALLY THE DECODING FOR THE LIGHTS

L1						16	8	4	2	1	
	6	12	14	22	23	X <sub>1</sub>	X <sub>2</sub>	y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	φ
	x	2	8	16		φ	φ	1	φ	0	30, 4, 20, 28
			1	x		1	φ	1	1	φ	31, 30,

Diagram 32

L1's decoding, too, is quite normal, except for one small point, and that is when we come to the y1 variable in row 1. 6 - 4 = 2 IS available to us, but not one of the other minterms will "go". In spite of this we do not try to transfer, neither do we place a dot over the y1 variable BECAUSE MINTERM-2 IS A PHI. AND WE ARE NOT COMPELLED TO CONSIDER IT IF WE DONT WISH TO DO SO. In this instance, we elect to consider it as a 0, and we can therefore say that 6 - 4 = 2 is NOT available, and proceed with the rest of the row. The final expression for L1 is

$$\begin{aligned} L1 &= y1.y3' + X1.y1.y2 \\ &= y1(y3' + X1.y2) \end{aligned}$$

L2						16	8	4	2	1	
	3	10	11	17	19	X <sub>1</sub>	X <sub>2</sub>	y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	φ
	x		8		16	φ	φ	0	1	1	27
		x	1			φ	1	0	1	φ	26, 27
				x	2	1	φ	0	φ	1	25, 27

Diagram 33

This is a very straightforward decoding, apart from a repeat of the situation above in row 1. Variable  $y_1$  gives  $3 + 4 = 7$  as a possible run, but as none of the other minterms will go, and minterm-7 is a phi, we choose to ignore it, thus making row 1 end up as an essential prime-implicant. The complete decoding for L2 gives

$$\begin{aligned} L2 &= y_1' \cdot y_2 \cdot y_3 + x_2 \cdot y_1' \cdot y_2 + x_1 \cdot y_1' \cdot y_3 \\ &= y_1' (y_2 \cdot y_3 + x_2 \cdot y_2 + x_1 \cdot y_3) \end{aligned}$$

L3				16	8	4	2	1		
5	13	15	21	$x_1$	$x_2$	$y_1$	$y_2$	$y_3$	$\phi$	
x	8		16	$\phi$	$\phi$	1	0	1	29	
	2	x		$\phi$	1	1	$\phi$	1	31, 29	

Diagram 34

Here again, in L3's decoding, we find  $y_1$  commencing a run with a phi, ie  $5 - 4 = 1$ , but none of the other covered minterms will go, so we elect to disregard it and carry on to make the row an essential prime-implicant. The final decoding for L3 is

$$\begin{aligned} L3 &= y_1 \cdot y_2' \cdot y_3 + x_2 \cdot y_1 \cdot y_3 \\ &= y_1 \cdot y_3 (y_2' + x_2) \end{aligned}$$

### ALL DONE! LET'S SEE WHAT OUR CIRCUIT LOOKS LIKE!

The complete network is shown below, with each output being kept separate, although some further economies would be possible if certain contacts common to two or more individual networks were combined. We shall learn how to produce multiple-output networks later on in this series.

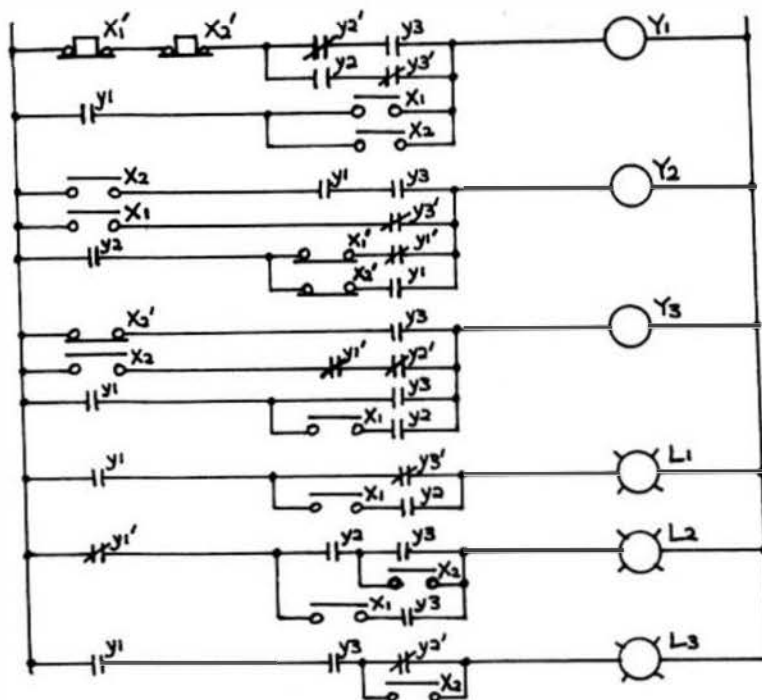


Diagram 35

Isn't that a complex piece of circuitry? Who'd have thought just a short while ago that a bunch of amateurs could design something like that? Of course, if one of us had decided on a different binary sequence for Gray-coding the state-diagram, (s)he'd end up with a completely different, but nevertheless functionally equivalent circuit! To really give it our best shot, however, we should also try decoding the zeroes (or non-available minterms) for each controlled device, and then complementing the resultant Boolean expression. We could then select the better network for each individual device, but that's just TOO much work right now.

This completes the basic procedure for the synthesis of sequential control circuits, but don't conclude from this that we've learned pretty well all there is to know. We've still barely covered a quarter of our journey, so next we're going to take an in-depth look at MERGING and GRAY-CODING, so that we'll better understand the processes we'll be using so often in circuits of this type.

### **SOME IDLE CHIT-CHAT**

Those of you who work with PLAs (Programmable Logic Arrays) which are programmed with a "Ladder" language, or even directly with Boolean expressions, should be able to make good use of what we've learned so far, especially the minimisation techniques. For the benefit of the uninitiated, a "ladder" language is based on our "ladder-like" network, and uses the symbols for NO and NC contacts for programming purposes.

Maybe some of you would be interested in writing a program to enable us to implement the design process on our computers, and thus eliminate the possibility of human error during the Gray-coding or decoding process. Just imagine, our very own CAE (Computer Aided Engineering) system!! Wow! I'd be MOST interested to know if you do any work done in this direction!!

Anyway, you've all worked so hard to keep up with me so far that I'm going to give you yet another test-free month. Unless, of course, any of you object to this!! Anyone here who's willing to risk the wrath of the rest of the party and INSIST on a test of some kind? No? I thought not ... so you can take time out to explore around for a while. It's even safe to go back to M'bul-yan territory now that we know how to 'discombobulate' old I-asku!!! See you back here next month.

... End of Mile 8

**EOF**

**FOR THOSE WHO NEED TO KNOW**

**68 MICRO  
JOURNAL™**



## The Macintosh™ Section

Reserved as

A place for your thoughts

And ours.....

## Mac-Watch

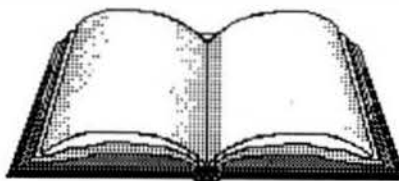
### THE WORD Processor

King James and New International Versions

The following is a review of a product that was a formidable effort on the part of its programmers-Bert Brown and Kent Ochel. It's an application, while selective and broad in scope, is very sophisticated in the manner in which it performs. And, as the vast array of other available applications are indispensable to certain users, such as word-processors, spread-sheet programs, accounting, CAD-CAM, etc., this program is one of the most, if not, next to the Bible, *the most valuable tool* any student of the Bible, who uses a personal computer, could own.

The Word processor allows the user to locate or work with any size of data block according to pre-set conditions (user defined). The user can search for any word, phrase, set of words, beginning, ending, or inside of other blocks, of the entire Bible, from Genesis to Revelation. All this at speeds that will astonish even the user of one of the faster spread-sheet applications. A search from one end to the other (Genesis to Revelation) for a single word can be accomplished in several seconds. Remember, the Bible contains over 31,000 verses or about 4.5 million characters.

When run on the Macintosh II (a 68020, 16MHz machine), it seems to be practically instantaneous! On the Macintosh Plus it is still very fast. For instance, I ran



it on the word "word" (Macintosh II) from Genesis to Revelation. There is 673 occurrences of "word" in the Bible, it found all 673, created an index of them, complete with each verse ready to be scrolled on the screen, with its complete text, and saved to disk in less than a minute and a half! I then had it search for the word "atomic" which is not in the Bible, it came back instantly and informed me that it was not in the Bible. A search for the word "Reverend", which only appears once, took about 3 seconds on a Macintosh Plus, and the Plus is about four to five times slower than the II. This speed is in part due to a companion application that can be ordered as a separate package, Instant Access, which is keyed to every word in the Bible. Without Instant Access the search would have been somewhat slower, but not a lot. However, for its low price, I would recommend that the user also consider Instant Access when ordering The Word processor.

One of its most valuable features is the ability to create selective indexes of selected words, phrases, verses or any subset of them.

#### Products

Application	System
The Word processor	MAC, IBM
KJV - NIV	Apple, C64
Greek Transliterator	Mac, IBM,
Hebrew "	MAC, IBM
Topics	All Listed
People	All Listed
Instant Access	MAC, IBM
Personal Commentary	MAC, IBM
ASCII Utility	MAC, IBM,

A hard disk is strongly recommended as the entire Bible with the support programs will demand practically over 5 megabytes of storage. While diskette use is slower, for many sessions diskette use is acceptable. However, the user should be aware of the disk swapping required.

This review will be based on the Macintosh version. The following Macintosh functions are supported: Switcher, text files created in most all the Macintosh word processors using the Cut, Copy or Paste functions. The ASCII program can redirect output that would normally go to a printer to a disk file (not received or reviewed and no reports.) Windows are fully supported (no expansion click box, which would have been useful on the Macintosh II or any other wide or larger



screen Macintosh.) The mouse, menus and editing functions are supported. The Macintosh interface is supported to the extent that any Macintosh user can work within the programs without a lot of documentation research.

The Word processor is the "home-base" program. The Greek and Hebrew transliterators, Instant Access, Personal Commentary, People and Topics are imbedded in the basic The Word processor and become an integral part of The Word processor thereafter (for those with limited memory-512K, the transliterators may be installed or removed as the users desires.)

As mentioned before The Word processor can directly access any word, group of words, etc., in the entire Bible. Permanent concordances of any subject is supported. All Bible text relating to any subject referenced can be printed, with the "Red Letter" words of the New Testament indicated by <...> delimiters. Functions of the Main Menu are as follows: Display Help Instructions, Set Options, Create Indexes or work with previously created indexes, print comments, and several other features to be referenced below.

A dialog box asks for the range of text you are interested in. This comes up in the pre-set option which is permanently saved, but can be changed for any particular session, or saved as a new permanent option. The range can be in the form of book to book, book and verse to book and verse, or any combination of either. Once the subject(s) you indicated are located they may be viewed and scrolled as desired, in either the forward or backward direction.

The Edit menu has all the standard Macintosh functions such as, Undo, Cut, Copy, Paste, Clear, Select All, Options...which allows the user to set a standard set-up and Change Mode to SRN (provided you have installed the transliterators.)

The Indexes menu allows, Open an existing Index, New Index, Delete an existing Index, Combine Indexes and Build an Index.

The Search function allows Find Text, Find SRNs, Go To, Go to Beginning of Range, Find Again, Go to End of Range. A total of 32 references may be entered for each search or find. Rules for searching are very comprehensive. Preceding the referenced with the "=" sign finds all references that end with the reference. References that have the "=" sign after the word indicate the reference is a prefix, example-word=, finds word and words, as well as any other whole word starting with "word". Typing "=word=" will find all words containing "word". Items displayed or found by the building of an Index may be sorted to Biblical sequence, with duplicate references removed. Index references may be individually inserted, skipped, all inserted, or selectively removed, as desired. The Build Index is a faster way to create a new index than the New Index function, bypassing the selective options indicated above.

The Comment menu is active if you have installed the Commentary option. The Transliterator menu allows the following: Display SRN Scripture, English Word Frequency, Transliterated Word Frequency and Display Dictionary. The results of menu functions can be merged into any existing index as the user desires.

#### **The Greek and Hebrew Transliterations**

These programs, when installed with the KJV, key with Strong's reference numbers (SRN.) Some Greek and Hebrew words do not directly translate across the languages. Many have been translated into the same English word. The Greek and Hebrew dictionaries contain the English definitions, the Greek or Hebrew transliteration, word origin and derivations. Also the user can find different English translations of Greek or Hebrew words and find the different words translated to the same English word. The Word Frequency function allows for this and displays the SRN number, the frequency of the word in question and the total number of occurrences. Strong's reference numbers are not available for the NIV. The Hebrew transliterator works the same as the

Greek version, but for the Old Testament with Hebrew reference numbers and a Hebrew dictionary.

The Display Dictionary function displays the SRN, the word associated with the SRN, derivation and definition. Indexes of Greek word contents can be created, and saved in the same format as regular indexes. By this process indexes can be merged or made interchangeable. These programs allow the average student to use them without any special knowledge of either language.

Hieretofore this type of reference work has been limited to extensive study and training on the part of the user. Now anyone can immediately study word usage in either language. A student of the Bible can refer to the original Greek for reference in the New Testament and the Hebrew for the Old Testament with confidence in these research tools.

Documentation is sufficient, but not overbearing. I found that within an hour or so I was able to use the programs without having to go to the books often. In addition, if disk space allows, a very comprehensive Help reference is available under the Apple menu. With Help installed, I put the books away.

#### **The ASCII Utility**

The ASCII Utility is a \$49 optional program that allows The WORD processor files to be printed to disk in ASCII form. They may then be opened, read, modified, and printed by most word processors including MacWrite. To see the value of this utility, the handling of files without it should be understood.

The WORD processor uses its own printer driver rather than the standard Apple printer drivers. Most windows have a Write box that, when clicked, causes the active window to be printed to any Apple ImageWriter or LaserPrinter compatible printer. Alternately, **Print Selection** from the File menu can be used to print whatever text has been selected. Printing on the image writer is *extremely* fast since the printers internal fonts are used in a "draft" mode. In my MacPlus system, the printer begins in about 2 seconds and a full screen of text

is printed within 10 seconds. The single—spaced output is quite legible, though not very dense. This arrangement is so fast that printing windows or selected text as you study (rather than waiting until you finish) is entirely practical and hardly interrupts your train of thought.

This printing arrangement is entirely satisfactory for study. A way is needed, however, to use output from The WORD processor with other applications such as desktop publishing and word processing programs. The normal Macintosh Copy function is helpful but it only works with selected windows. For example, you *can* copy a selection of text to the clipboard but you *cannot* copy the contents of an English Language Frequency window to the clipboard. Since the normal printer driver arrangement is not used, programs like GLUE cannot be used to capture WORD processor images.

You understand then, that the ASCII Utility is absolutely essential to any serious user of the WORD processor. When this utility is installed, Write Options from the File menu allows you to add the selected material to an existing ASCII file, open a new file, or "write" to the printer. Creation of ASCII files is reasonable fast. Transferring a 21K index into a ASCII file took about 55 seconds on my system. The resulting file can be read by MacWrite and any application that understands MacWrite files.

#### The Topics Product

The WORD processor's ability to create *custom* indexes is one of its more powerful features. Often, however, the subject you're interested in will be a common one which is of interest to many Bible scholars. If this is the case with you, you will find the topics index (a \$49 option) an invaluable aid. This index contains over 200 topics from "adultery" to "zeal."

This index is not merely a list of verses containing a specific word, but rather those verses that deal with the selected *topic*. Obviously this required a great judgement on the part of those that prepared the index and you may not always agree with their conclusions. For example, whether you accept that the 32 references given under "Baptism of the Holy Spirit" really relate to that topic will depend on what you think the baptism of the Holy Spirit really is.

#### The People Product

The People Product is a collection of indexes that cross reference over 140 people mentioned most often in the Bible. From "Aaron" to "Zerubbabel," this \$49 option provides a useful tool to help the user better get to know famous characters of the Old and New Testaments. Like the Topics Product, these indexes are based on context, not use of the person's name. Therefore, verses about a character which do not specifically mention the character's name are included. These indexes are not just a sampling of passages, but are quite *comprehensive*. For example, 588 verses are listed for the Apostle Paul and 71 are listed for Mary, the mother of Jesus.

The Topics and People indexes are made far more useful by the powerful integration of features in The WORD processor. For example, the first screen displayed when you open an index lists all the verses related to the selected topic or person. Double click the number of any verse and the text of that verse is displayed. Type COMMAND + 'D' and instantly see all Strong's Reference numbers associated with that verse. Double click any listed reference number and instantly the original Greek or Hebrew word and its meaning is presented (provided you have the optional transliterations). This brief description does not even scratch the surface of the power and flexibility of the WORD processor's handling of indexes.

#### Conclusion

While this particular set of programs may be of limited interest to some readers, it is without doubt, one of the most exhaustive and complete software packages I have ever tested or used! We have in our Macintosh library hundreds of titles, ranging from those that are utterly useless to those that we cannot do without. If you are a student of the Bible, Old or New Testament, then this software is a must! *I cannot recommend it too highly.*

You can secure additional information from the vendors below, or call or write S.E. Media (see advertising this issue) to order direct:

Bible Research Systems  
2013 Wells Branch Parkway #304  
Austin, Texas 78728  
(512) 251-7541

A staff review.

EOF

FOR THOSE WHO NEED TO KNOW

68 MICRO  
JOURNAL™

# FORTH

## A Tutorial Series

By: R. D. Lurie  
9 Linda Street  
Leominster, MA 01543

### WRITING AND DEBUGGING A FORTH PROGRAM

In a recent letter, Wilson Federici suggested that I try to show how one goes about writing and debugging a FORTH program. I thought that he had a good idea, and that I would try to follow his suggestion; particularly since I have never seen but one other attempt to do that. Actually, it is easier to write a program and debug it than it is to show how it was done, but I will give it a try.

In my last column, I discussed how I had written a set of definitions which would allow a FORTH application to call a ROM routine, which, in turn, would call the SWTP CALC-1 math board. I promised to show how to use it in a specific application, along with an easy way to call for a long list of input data. I decided to present both of those objectives and show how I went about writing and debugging the definitions, all in this column.

At the same time, I may answer some of your other questions, such as why I jump around with screen numbers, so that they do not always fall into an obvious sequence.

#### GETTING STARTED

The answer to that question is that I usually start a development project with a fresh disk. By assigning a whole disk to one application, I don't have to worry about accidentally overwriting a screen which I want to save. Since I have SSDD 80-track drives, I have 355 screens, or 363,520 bytes, of virtual memory to play around in! I use screen #0 as a crude table of contents to keep track of where major segments of the program are stored, and I combine these blocks, once I think that I have a reasonably functional program written.

I have no problem loading these scattered groups of screens whenever I want to test the current state of the program, because I use a

loading screen to call all of the screens in the proper sequence. By using a loading screen, instead of —> to call the next screen, I can skip around during loading, and, of course, FORTH couldn't care less the order in which it reads a disk.

I will have to make a few assumptions during this presentation in order not to waste a lot of space. I will assume that you have a copy of my previous column, so I will just refer to those screens without reprinting them here. I will also assume that I can refer to a definition, such as  $X > M$ , without giving more than a passing reference to its meaning. I really don't expect that you will have any trouble following me.

#### CALCULATING THE STANDARD DEVIATION

I am sure that all of you have heard the words "standard deviation," but not all of you will know what it means. In a somewhat simplistic manner, I will try to describe its meaning and how it is calculated, hopefully without offending any mathematicians who might read this.

The standard deviation is the probable range, or limits, within which two-thirds of a group of numbers will fall. These numbers are centered around the mean, so that one-third of the numbers will probably be less than the mean, but not less than the lower limit, and one-third of the numbers will probably be greater than the mean, but not more than the upper limit. The mean value is found by taking the sum of all of the numbers within a group and dividing that sum by the count of the numbers which went into that group; this is the result which most of us have come to call the "average" of the set of numbers.

Since the standard deviation which we will calculate represents only the probability, and not the hard fact of a situation, it is commonly referred to as sigma, which is the square root of the difference between the square of the mean of the numbers and the mean of the numbers, squared. This may be a little more clear if I use a pseudo-equation:

$$\text{sigma} = \text{square root} ( (\text{mean} (X^{**2})) - ( (\text{mean}(X))^{**2} ) )$$

The sigma can be used to estimate the range of a given set of numbers for a stated confidence limit. For example, it is possible to say that the data from a reasonably sized experiment can be used to tell me the range of any new numbers (if I continue to run the same experiment) to any given confidence limit. As an example, drug testing is done in such a way that the FDA can be confident that 95%, or 99%, or 99.9%, or 99.9999% of the population will react in an expected way to a new drug treatment. Notice that there is no way that the math will let you be 100% sure of anything. However, I have used "confidence limits" in over 20 years of engineering experience, and never had an exception show up.

I don't want to spend any more time on discussing why you might want to calculate sigma and confidence limits, because I will run out of room for FORTH. Instead, if you want more information on this very interesting subject, check with your public library. They are sure to have several books on the subject, ranging from beginner's texts to stuff to boggle the mind of a math PhD!

### THE LOADING SCREEN

Screen #100 is typical of the type of loading screen which I use while developing a program. However, in this particular case, the illustration shows the end result, but it will serve to demonstrate the main points. I will show its evolution as we go along.

I normally use line #0 to describe the program, or segment of the program, which this loading screen is expected to cover. As a result, I can simply list the first lines of the various loading screens in order to know where to look for a particular segment of a program. As the program gets larger, or at the end of a major portion of the development cycle, I will start to build more elaborate and descriptive loading screens. In this case, the program for calculating the sigma value is short enough that this relatively simple loading screen is enough.

The first line also contains a "date stamp," which I like to include with all of my screens (and each individual definition). The purpose of the date stamp is simply to make it easy to know when I made the last change in a screen (or definition). Every time I make a change in a definition covered by a particular loading screen, I make a point to update the date stamp on the loading screen, also. I have written an addition to my FORTH editor to make this easy to do, and I will describe it in another column.

Line #2 is the beginning of the list of screens to be loaded. FORTH will attempt to execute any words found outside of a definition while loading a screen; that is why comments must be specifically marked in order to keep them from confusing the compiler. Therefore, a phrase like 1 LOAD will be executed, if it is outside of a : ... ; pair. Likewise, any additional such phrases will also be executed as they are encountered.

A loading screen can call another loading screen, ad infinitum, so the phrase 1 LOAD can be used to load the set of calculator control screens which were described in the last column. I think that this is the easiest way to gather a number of previously debugged screens into an application currently being developed. In fact, I have a disk of over 200 screens, which I call my "utility disk" which I use as a source for many of the common chores; pretty much like the "library disk" used by C programmers. I copy the reusable screens from this utility disk onto a segment of the working disk, and then I compile these common definitions into my working program with one or more loading screens in the way I have used, here, the phrase 1 LOAD.

The remainder of line #2 happens to refer only to screens associated specifically with this program, so I will not spend much more time on it.

There is no need for all of the calls to LOAD the various screens to be on the same line. It just happened to be convenient to do so, in this example. In fact, as I think about it, it probably would be more convenient to have each LOAD command on a separate line, simply to make it easier to change each one without affecting another. Suit yourself and do which ever one is the more convenient for you!

As a last comment on screen #100, I would like to discuss line #4. Whenever there is enough room left on the loading screen, I like to include a short statement of the purpose of the program or a condensation of the algorithm. This helps me to keep in mind the real reason for developing an application, and cuts down on any tendency to stray into unproductive side issues. Again, this is a "suit yourself" issue; use the suggestion if you find it helpful; otherwise, forget it.

## PROGRAM DATA

Screen #101 is exactly the same as presented last time. It is the list of data which will be used to test this program, since I already know what the result should be. Obviously, this is one of the best ways to catch any bugs in a math program; if you don't get the expected answer, you know that there has to be a bug somewhere! The data screen can be located anywhere, but I chose this as a convenient place to put it. DONT TRY TO LOAD SCREEN #102! This mistake will produce from 1 to 33 error signals, depending on just how your particular FORTH tries to process screens during loading. This is why 102 LOAD is the first phrase in screen #100 to refer to this group of screens. In fact, a good argument for using a loading screen is that it could make it less likely for you to load an incorrect screen. Later on, I'll discuss, in excruciating detail, how to use screen #101.

The first step in developing this program was to be sure that the data screen could be read correctly. In order to do that, I needed to know how many data items there would be. This meant that I had to read the first number on the disk; in this case, line #2 of screen #101. The easiest way to do this was to bring screen #101 into the file buffer and, then, to treat the buffer as an array. In this way, I could use NUMBER to fetch the count to the top of the Data Stack.

Therefore, screen #103 was written. The purpose of this screen was quite limited; initially, I only wanted to fetch the screen and move the count to the Data Stack. In order to fetch the data screen, I only needed to enter, from the keyboard, the phrase

```
101 BLOCK<CR>
```

In order to copy screen #101 from the disk to the file buffer. Furthermore, the address of the first byte of the buffer was left on the top of the Data Stack, ready for further use. This sets up the Data Stack in readiness for the execution of SIGMA.

Line #1 of screen #103 is a little bit of cheating, since I knew that I was going to need extra copies of the buffer address later on. I went ahead and copied the address, here, for later use. Also, I knew that I would need to know the address of the buffer if I had to debug this definition of SIGMA at this point. So, just consider the DUP as a little bit of insurance.

Line #2 makes the conversion from the ASCII string into the number stored on top of the Data

Stack. The first byte of the number-string to be converted begins at the 64th byte of the buffer. Remember that NUMBER requires a pointer to the FIRST BYTE OF THE STRING MINUS ONE! Therefore, we want to point to the string by adding 63, not 64, to the address of the first byte of the buffer.

NUMBER produces a 32-bit result, and we can only use a 16-bit integer with a DO ... LOOP, which is the way we plan to use the count. FORTH always has the higher 16-bits of a 32-bit number as the top-most integer, so a simple DROP will convert the product of NUMBER into an integer which we can conveniently use.

I decided that I would save myself some trouble with stack management by keeping the count in a VARIABLE, rather than juggling it with the addresses which would be kept on the Data Stack. By storing the count into \_COUNT, I moved it safely out of the way, and could rely on it always being conveniently available whenever it was needed.

Notice that I keep using the word "convenient." I think that this is a very important notion. If program development were not convenient, I think that I would lose one of the major advantages of FORTH. If I had to sweat and strain to develop a program, I would waste a lot of time and effort. Why not keep things as simple and convenient as possible during program development? If necessary, after the program is actually running and reasonably debugged, then go back and make changes in the program that were put in there for development convenience; but don't let the search for program "elegance" distract you from the main task of writing a functional program.

Finally, the ; in line #4 is necessary in order for the definition to be properly compiled. By being on a line by itself, the ; does not have to be erased and rewritten as new words are added to the definition. Therefore, I have honored the concept of "maximum programmer convenience."

This definition can be tested by using the following loading command:

```
1 LOAD 103 LOAD<CR>
```

You can also replace line #2 of screen #100, as I did. The command:

```
100 LOAD<CR>
```

would then cause the group of screens controlled by screen #1 to be compiled, followed by the preliminary definition of SIGMA from screen #103.

When you are satisfied with the program, to date, you can clear the dictionary by typing:



FORGET MC<CR>

(refer to screen #1 for the definition of MC ).  
Execution of the command:

101 BLOCK SIGMA<CR>

causes the expected results—32 stored in \_COUNT and the buffer address on top of the Data Stack.

Now that we know that we can properly read the count from the data file, we are ready to try reading the actual data items. We do this by adding 128 to the address already on the Data Stack, because the first data byte is at the beginning of line #2 on screen #101. This pointer is calculated in line #4 of screen #104. Notice that I was able to add this line to the screen without changing any of the previous lines; this is what I meant when I talked about programming convenience.

At this point, I realized that I was going to generate enough code that it would not fit into just one screen. Therefore, I decided to factor the definition of SIGMA by means of the sub-definition (SIGMA) . I could have force-fed the definition of SIGMA with the additional code I needed right now, but there was no point in doing that, since it was obvious that factoring would eventually be necessary. Besides, good factoring is the essence of good FORTH code. Well, I must admit that my code may not always be good FORTH, but it certainly is factored!

Screen #105 contains the definition of (SIGMA) . You may ask how we can put the definition of a factor after the definition of the root word. The answer is that, since we are using a loading screen, instead of → to load multiple screens, it makes no difference what might be the numerical order of screens. For testing at this stage, our loading screen will contain:

1 LOAD 102 LOAD 105 LOAD 104 LOAD

Since I had planned to read and process the input data through a DO ... LOOP , I must set it up, now. Line #1 of screen 105 fetches the count from \_COUNT and sets the limits of the DO ... LOOP . At this stage, I am only going to be sure that the data are read correctly by @INPUT . Remember that @INPUT leaves a pointer to the next data item on top of the Data Stack, so that the address calculated in line #4 of screen #104 is the only pointer which we need to calculate, explicitly.

Lines #3 and #4 are somewhat redundant, but I wanted to make sure that the CALC-1 math board was actually doing exactly what I expected it to. Line #3 is all that is really necessary to

display the input data after initial processing by the calculator, but line #4 was emotionally satisfying.

Line #5 of screen #105 simply closed the DO ... LOOP .

Testing the program at this point demonstrated that everything was working as expected. Therefore, I was ready to proceed with the actual computations, which were based on the RPN algorithm shown in Figure 1.

1. Read and enter a data item.
2. Save a copy of the input data for later use
3. Add the new data item to the running total.
4. Recover the copy of the input data and square it.
5. Add this squared value to the running total of squares.
6. Repeat steps 1-5 until all of the input data have been processed.
7. After all of the data have been entered, calculate the mean of the sum of the input data and square this mean.
8. Subtract this number from the mean of the sum of the squared input.
9. Calculate the square root of this difference and report it.

Figure 1. The algorithm for calculating the standard deviation, SIGMA .

Examination of screen #106 shows that the first 4 lines are identical to those of screen #105. Lines #4-5 save a copy of the input data in the memory register of CALC-1, and then add the input data to the running total, which will be used later.

Line #6 is included as an indication that everything is proceeding as it should, or as a debugging aid. SHOW displays all of the CALC-1 registers, and KEY DROP causes everything to pause until I press any key. This gives me time to study the data displayed by SHOW before it has a chance to scroll by and be lost to view.

This time, the proper command for testing is:

1 LOAD 102 LOAD 106 LOAD 104 LOAD

You could put the command FORGET MC into screen #100 as line # 1, but I chose to keep it separate. FORGET MC would produce an error and prevent compiling whenever you tried to use 100 LOAD from a cold start. I count this as an unnecessary aggravation and prefer to keep FORGET MC as a separate command. The worse that can happen by not using it is that I get a lot of warning messages, but I still get compilation! Now that I know that the basics of the program work properly, I am ready to add the remainder of the calculations. Screen #107 is the new version of (SIGMA) . The first six lines are unchanged from screen #106, and the lines #6-8



simply complete the operations necessary for steps 1-6 of the algorithm in Figure 1. Lines #9-10 of screen #107 were simply pushed down from lines #6-7 of screen #106; this was easy to do with the editor supplied with F9.

At this point, I encountered my first serious bugs! There are two of them. The first one is that I cannot possibly get the correct answer for the standard deviation, because I forgot to clear the CALC-1 internal registers before making the first addition. If I fail to clear these registers, then I don't have an initial zero base for the first additions. The CLEAR-CALC command in line #1 of screen #110 does this initialization, and it must be done as the first operation of the program.

The CLEAR-CALC command forces CALC-1 into floating point mode. This may be acceptable for small numbers, but we can quickly get an overflow error if any number anywhere during the calculations should grow larger than 99,999,999. CALC-1 gives no useful overflow warning, nor does it automatically switch into scientific notation, so we must explicitly switch into scientific notation mode, before doing any calculations. The TOGGLE-MODE command takes care of this problem.

The insertion of line #1 into screen #108 is the only change from screen #104. The new version of line #2 of screen #100 is:

```
1 LOAD 102 LOAD 107 LOAD 108 LOAD
```

The insertion of line #1 into screen #108 fixed all of the bugs, so far, so I decided to add the rest of the calculations to the definition of (SIGMA). This is shown in screen #109, lines #10-15. I also made some other, minor, changes to the definition of (SIGMA), one of which was simply to comment out line #3, since I knew that this part of the program was working correctly. By removing this line with a \, but not erasing it from the screen, I could, in effect, reinsert this line at any time by erasing the \ and recompiling.

The other minor change was to separate the two pointer designations into "adr1" for the start of the buffer and "adr2" for the start of the data. I had to do this in order to get a better control over the root definition SIGMA. Possibly, I should call this a major change in (SIGMA), but I think that it is really more of a major change in SIGMA. In any case, the argument is moot, since the change had to be made in both definitions. However, since the two pointers are both carried on the Data Stack, the names "adr1" and "adr2" mean nothing to FORTH; they are just used for programming convenience (there's that word, again!).

A change in the definition of (SIGMA) requires a corresponding change in its root definition SIGMA. Therefore, we can't test the new version of (SIGMA) until we have made the changes shown in screen #110. This version of SIGMA now provides the two pointers, "adr1" and "adr2." OOPS! Where did that \*\*\*INPUT ERROR\*\*\* come from?!? Removing the \ from line #3 of screen #109 did not provide an answer, so I had to dig deeper.

Since the error was with the input data, I knew that SHOW would not tell me anything useful. The only thing left, obviously, was DEBUG. You may well ask why I did not immediately resort to DEBUG, and my only answer is "too lazy." The use of DEBUG usually results in so much data being dumped out so fast, that it is sometimes hard to spot just where the program went wrong. Therefore, I bowed to the inevitable and entered the following command string:

```
DEBUG (SIGMA) 101 BLOCK SIGMA<CR>
```

which produced the result shown in Figure 2, the last twelve lines of output to the display. Of course, there is the problem! The DUP on line #11 of screen #109 is not copying the expected pointer! It is simply duplicating the pointer to the beginning of the disk buffer, and not to the pointer to the "count" bytes. (By the way, do you see the other error, which I will point out a little later?)

```
B7FA B91A      ANSWER4.80000000 E03
B7FA B91A      >MEM
B7FA B91A      PLUS
B7FA B91A      X><M
B7FA B91A      SQUARE
B7FA B91A      PLUS
B7FA B91A      MEMO
B7FA B91A      (LOOP)
B7FA B91A      DROP
B7FA           DUP
B7FA B7FA      @INPUT
***INPUT ERROR***
```

Figure 2. The end of the output resulting from the command:

```
DEBUG (SIGMA) 101 BLOCK SIGMA<CR>
```

About this time, it occurred to me that the 63 on line #3 of screen #110 was a little too obscure in meaning. Therefore, when I wrote screen #111, I decided that I should change this to 64, so that its meaning would be more clear the next time I looked at the listing of this program. (Does this mean that the "scientific method" is what you do while you are waiting for inspiration?)

As soon as I made this change, like the proverbial light bulb, I recognized the problem with the previous version of the program. I had forgotten that @INPUT had to have a pointer to the first byte

of a string convertible by NUMBER . Since \$B7FA pointed to the start of the disk buffer, it could not satisfy this requirement; no wonder I got the error message! Once I recognized the problem, the solution was simple.

In order to save a calculation later, I decided to make a copy of the pointer to the count byte string, which I am now calling "adr1." This was done with the DUP in line #3 of screen #111.

The phrase 1- NUMBER completes the operation which had been on the single line #3 of screen #110. Now the Data Stack contains the original pointer to the buffer, a pointer into the buffer, a pointer to the count byte string, and the count integer, in that order. The count is stored into the \_COUNT variable in line #5. This causes the Data Stack to contain only the two pointers; albeit, in the reverse order to what is needed. However, this is easily fixed by the SWAP in line #6. Now, line #7 of screen #111 is the same as line #5 of screen #110, and we can add 128 to the pointer to get the "adr2" required by (SIGMA) . The screen is beginning to look kind of messy, but the definition of SIGMA , itself, is still pretty clean, if you remove all of the comments. Therefore, I think that I will let the messy look ride, for now.

Ok, so let's test again, with line #2 of screen #100 becoming:

```
1 LOAD 102 LOAD 109 LOAD 111 LOAD
```

Oh, blast it! The program worked, but I got the wrong answer! Well, nothing for it, but I must now debug the math sequence. Therefore, I copied screen #109 to screen #112 and added all of the SHOW commands. Screen #100 was changed to:

```
1 LOAD 102 LOAD 112 LOAD 111 LOAD
```

in an effort to find the math error.

Hoist on my own petard, again! The @INPUT has the wrong pointer. Actually, the problem is already fixed; I just failed to let it happen. I forgot to DROP the unnecessary pointer placed on the Data Stack by @INPUT when it finished reading the count string the previous time. The addition of DROP in line #13 of screen #113 would kill this bug, and the loading line now became:

```
1 LOAD 102 LOAD 113 LOAD 111 LOAD
```

At last! I got the right answer this time, and the algorithm is finished, so there is nothing left to do, but tidy the screens a little.

There is certainly no question that screen #113 is in too much of a mess to leave it that way, if I ever expect to understand it a week later. Since the algorithm really has two phases:

- (1) reading and calculating the running totals, and
- (2) calculating the square root of the differences,

that would certainly appear to be the logical place to split the screen. Therefore, I rewrote screen #113 into screens #114 and #115, calling the two definitions (SIGMA1) and (SIGMA2) . Of course, this also meant that a new version of SIGMA was required, and this became screen #116. When this was done, the new version of the loading screen became the one now shown in screen #100.

## CONCLUSION

I hope that some of you found this effort useful, and the rest of you were not bored, too much. If any of you want me to do other columns like this one, please let me know. Otherwise, I will stick to presenting more-or-less finished programs, and not go into so much exhaustive detail on their development.

On the other hand, please let me know if I have been spending too much time with basic detail. I will try to be guided by the kind of feedback, if any, that I get.

## FEEDBACK

Speaking of feedback, I would like to get more! Letters and telephone calls are welcome, but you can also reach me on DELPHI as RDLURIE and CompuServe as ID# 72667,3437. I check into each one at least twice per month, so I can get EMAIL on either. Let me hear from you about what you want, your gripes, news, etc.

```

SCR #100
0 \ Standard deviation loading screen          \ RDL 09/01/87
1
2 1 LOAD 102 LOAD 114 LOAD 115 LOAD 116 LOAD
3
4 \ SIGMA = square root ( (mean (X**2)) - ( (mean (X))**2 ) )

SCR #101
0 \ DATA                                     \ RDL 09/01/87
1 32
2 5500 5500 5500 5500 5200 5200 5200 5200 5000 5000 5000 4800 4700
3 5000 4600 4800 4500 4400 4700 4700 4600 4400 4600 4500 450
4 0 4500 4500 4700 4800 4800 4800

SCR #102
0 \ CONSTANTS, etc                          \ RDL 09/01/87
1
2 VARIABLE _COUNT

SCR #103
0 : SIGMA ( adr - )                          \ RDL 09/01/87
1 { adr } DUP                                \ copy of block address
2 63 + NUMBER                                \ fetch the data "count"
3 DROP _COUNT !                             \ save the count
4 ;

SCR #104
0 : SIGMA ( adr - )                          \ RDL 09/01/87
1 { adr } DUP                                \ copy of block address
2 63 + NUMBER                                \ fetch the data "count"
3 DROP _COUNT !                             \ save the count
4 { adr } 128 +                              \ point to input data
5 (SIGMA) ;                                  \ do calculations

SCR #105
0 : (SIGMA) ( adr - )                        \ RDL 09/02/87
1 _COUNT @ 0 DO                             \ set loop limits
2 @INPUT
3 CR .ANSWER                                  \ display initial input
4 SHOW
5 LOOP ;

SCR #106
0 : (SIGMA) ( adr - )                        \ RDL 09/02/87
1 _COUNT @ 0 DO                             \ set loop limits
2 @INPUT
3 CR .ANSWER                                  \ display initial input
4 >MEM                                         \ save copy of input
5 PLUS                                         \ running total of input
6 SHOW KEY DROP
7 LOOP ;

```

```

SCR #107
0 : (SIGMA) ( adr - )                        \ RDL 09/02/87
1 _COUNT @ 0 DO                             \ set loop limits
2 @INPUT
3 CR .ANSWER                                  \ display initial input
4 >MEM                                         \ save copy of input
5 PLUS                                         \ running total of input
6 X><M                                         \ exchange total & input
7 SQUARE PLUS                                \ running sum of input**2
8 MEMD                                         \ ready for next operation
9 SHOW KEY DROP
10 LOOP ;

SCR #108
0 : SIGMA ( adr - )                          \ RDL 09/02/87
1 CLEAR-CALC TOGGLE-MODE                     \ initialize calculator
2 { adr } DUP                                 \ copy of block address
3 63 + NUMBER                                \ fetch the data "count"
4 DROP _COUNT !                             \ save the count
5 { adr } 128 +                              \ point to input data
6 (SIGMA) ;                                  \ do calculations

SCR #109
0 : (SIGMA) ( adr1 adr2 - )                  \ RDL 09/02/87
1 _COUNT @ 0 DO                             \ set loop limits
2 { adr2 } @INPUT
3 \ CR .ANSWER                                \ display initial input
4 >MEM                                         \ save copy of input
5 PLUS                                         \ running total of input
6 X><M                                         \ exchange total & input
7 SQUARE PLUS                                \ running sum of input**2
8 MEMD                                         \ ready for next operation
9 LOOP
10 { adr2 } DROP
11 { adr1 } DUP
12 @INPUT DIVIDE SQUARE                       \ (mean of sum of input)**2
13 X><Y                                         \ swap stack contents
14 @INPUT DIVIDE                             \ mean of sum of (input**2)
15 X><Y MINUS SQUARE-ROOT .ANSWER ;

SCR #110
0 : SIGMA ( adr - )                          \ RDL 09/02/87
1 CLEAR-CALC TOGGLE-MODE                     \ initialize calculator
2 { adr } DUP DUP ( = adr1 )                 \ copies of block address
3 63 + NUMBER                                \ fetch the data "count"
4 DROP _COUNT !                             \ save the count
5 { adr } 128 + ( = adr2 )                   \ point to input data
6 (SIGMA) ;                                  \ do calculations

```

```

SCR #111
0 : SIGMA ( adr - ) \ RDL 09/02/87
1 CLEAR-CAIC TOGGLE-MODE \ initialize calculator
2 ( adr ) DUP \ copy of block address
3 64 + DUP ( = adr1 ) \ point to "count"
4 1- NUMBER \ fetch the data "count"
5 DROP _COUNT ! \ save the count
6 ( adr ) ( adr1 ) SWAP
7 ( adr ) 128 + ( = adr2 ) \ point to input data
8 (SIGMA) ; \ do calculations

```

```

SCR #112
0 : (SIGMA) ( adr1 adr2 - ) \ RDL 09/02/87
1 _COUNT @ 0 DO \ set loop limits
2 ( adr2 ) @INPUT
3 >MEM \ save copy of input
4 PLUS \ running total of input
5 X>M \ exchange total & input
6 SQUARE PLUS \ running sum of input**2
7 MEMD \ ready for next operation
8 LOOP
9 ( adr2 ) DROP
10 ( adr1 ) DUP SHOW
11 @INPUT DIVIDE SQUARE \ (mean of sum of input)**2
12 SHOW
13 X>Y SHOW \ swap stack contents
14 @INPUT SHOW DIVIDE SHOW \ mean of sum of (input**2)
15 X>Y MINUS SHOW SQUARE-ROOT .ANSWER ;

```

```

SCR #113
0 : (SIGMA) ( adr1 adr2 - ) \ RDL 09/02/87
1 _COUNT @ 0 DO \ set loop limits
2 ( adr2 ) @INPUT
3 >MEM \ save copy of input
4 PLUS \ running total of input
5 X>M \ exchange total & input
6 SQUARE PLUS \ running sum of input**2
7 MEMD \ ready for next operation
8 LOOP
9 ( adr2 ) DROP
10 ( adr1 ) DUP SHOW
11 @INPUT DIVIDE SQUARE \ (mean of sum of input)**2
12 SHOW
13 X>Y SHOW ( adr1+3 ) DROP \ swap stack contents
14 @INPUT SHOW DIVIDE SHOW \ mean of sum of (input**2)
15 X>Y MINUS SHOW SQUARE-ROOT .ANSWER ;

```

```

SCR #114
0 : (SIGMA2) ( adr1 - ) \ RDL 09/02/87
1 ( adr1 ) DUP
2 @INPUT ( adr1+3 ) DROP
3 DIVIDE
4 SQUARE
5 X>Y
6 @INPUT ( adr1+3 ) DROP
7 DIVIDE
8 X>Y MINUS
9 SQUARE-ROOT
10 CR CR .ANSWER CR ;

```

```

SCR #115
0 : (SIGMA1) ( adr2 - adr2+x ) \ RDL 09/02/87
1 _COUNT @ 0 DO \ set loop limits
2 ( adr2 ) @INPUT
3 CR .ANSWER \ display input
4 >MEM \ save copy of input
5 PLUS \ running total of input
6 X>M \ exchange total & input
7 SQUARE PLUS \ running sum of input**2
8 MEMD \ ready for next
9 \ operation
10 LOOP ;

```

```

SCR #116
0 : SIGMA ( adr - ) \ RDL 09/02/87
1 CLEAR-CAIC TOGGLE-MODE \ initialize calculator
2 ( adr ) DUP \ copy of block address
3 64 + DUP ( = adr1 ) \ point to "count"
4 1- NUMBER \ fetch the data "count"
5 DROP _COUNT ! \ save the count
6 ( adr ) ( adr1 ) SWAP
7 ( adr ) 128 + ( = adr2 ) \ point to input data
8 (SIGMA1) \ do summations
9 ( adr2+x ) DROP \ tidy Data Stack
10 (SIGMA2) ; \ finish calculations

```

EOF

FOR THOSE WHO NEED TO KNOW

68 MICRO  
JOURNAL™

# Bit-Bucket



By: All of us

*"Contribute Nothing - Expect Nothing", DMW '86*

*microware*

MICROWARE SYSTEMS CORPORATION  
1900 N.W. 114th Street  
Des Moines, Iowa 50322

Phone: 515-224-1889  
Telex: 916-330-2638  
FAX: 515-224-1302

FOR MORE INFORMATION CONTACT:  
Mr. Andy Ball  
Vice President, Marketing  
Microware Systems Corporation  
1900 NW 114th Street  
Des Moines, Iowa 50322  
515-224-1929

FOR IMMEDIATE RELEASE - MARCH 4, 1988

## MICROWARE RELEASES OS-9 CROSS C COMPILER FOR HEWLETT PACKARD SERIES 300 WORKSTATIONS

DES MOINES, Iowa. Microware Systems Corporation proudly announces a new OS-9/XCC Cross C Compiler for Hewlett Packard Series 300 Workstations. HP Series 300 users can now assemble, link and compile OS-9/68000 C programs under the HP/UX operating system for real-time operation on OS-9 based systems.

Microware supports an efficient, high performance OS-9 C Compiler that conforms to the Kernighan and Ritchie standard. OS-9/XCC also provides extensive Berkeley 4.2 oriented libraries, complete with standard I/O and MATH routines, to maximize portability between the HP/UX and OS-9 operating systems. Users are able to develop new or relating HP/UX resident C programs with OS-9/XCC 68000 system components: C Executive, Preprocessor, One-Pass C Compiler, Optimiser, Utilities, Macro Assembler and linker for real-time OS-9 execution. OS-9/XCC programs compiled on HP 100 Series Workstations produce position independent, reentrant, compact executable object code for the OS-9 operating system. Users also have the option to output 68000 assembler source, linker, input or executable binary files.

The OS-9 Operating System is a real-time, multi-user and multi-tasking system for computers based on the entire family of Motorola 68000 microprocessors. OS-9 uses a independent modular architecture to support a host of file managers, high level languages and debuggers which are used world-wide for industrial and scientific applications. And since OS-9 is compact, both the operating system and executable C object files are completely ROMable for dedicated target systems.

Powerful communication packages are available from Microware to facilitate hardware links between OS-9 and HP Series 300 Workstations. The OS-9/ESP Ethernet Support Package provides Ethernet TCP/IP communications, supporting both PTP and Telnet, to facilitate file transfer and remote login to both host and target systems. Microware's COM Communication Package provides high speed serial communications to and from OS-9 systems and the HP Series 300 Workstation. Both packages assure high performance and compatibility for the transfer of C programs written with OS-9/XCC.

OS-9/XCC is distributed on standard HP Series 300 cartridge tape for easy installation. OS-9/XCC can be purchased by contacting Microware or an authorized Microware distributor.

Founded in 1977, Microware System Corporation specializes in the development of advanced operating systems and programming languages that has been licensed to manufacturers world-wide for use in industrial, scientific and consumer products. Last year Sony and Philips announced the OS-9 Operating System as the foundation for Compact Disc-Interactive (CD-I) New Media Technology. Microware offices are located in Des Moines, Iowa, Santa Clara, California and Tokyo, Japan with field representatives worldwide.

**MICRONICS**  
RESEARCH CORP.

Microcomputers, Hardware and Software  
GEMIX® Sales, Service and Support

33383 LYNN AVENUE,  
ABBOTSFORD,  
BRITISH COLUMBIA,  
CANADA, V2S 1E2

Dear Don,

It's been a long, long time since I wrote XBASIS XPLANATIONS, but as a result of a recent phone call from a reader I'm persuaded to write occasionally about any BASIC programming tips that I think might be useful. I don't intend to get into anything major, such as writing a STARTREK or GALACTIC SHOOT-EM-UP, but will merely discuss various techniques from time to time.

For instance, how about random numbers? Everyone, of course, knows how to generate a random number, let's say in the range 1 through 52.

```
100 J% = RND(0) * 52 + 1
```

This will do the trick nicely. Each time line 100 is executed it'll produce a random number in the required range, which may, or may not, be the same as one already produced. Which causes problems if we want to produce a sequence of numbers from this range with no two numbers the same. Suppose, as an example, we were writing a card-game program of some kind, and we wished to shuffle a deck of cards by randomising the integer sequence 1 through 52. What now? I've seen a program such as the following which does the trick:

```
100 C%(1) = RND(0) * 52 + 1: FOR I% = 2 TO 52
110 J% = RND(0) * 52 + 1: FOR K% = 1 TO I% - 1
120 IF C%(K%) = J% GOTO 110
130 NEXT K%: C%(I%) = J%: NEXT I%
```

Let's examine this little fellow! Line 100 begins by selecting a random card-number for Card-1, let's say it's 23, and then goes into an I%-loop to choose the remaining 51 cards. Beginning with Card-2, line 110 chooses another random-number, and then goes into a K%-loop to check this number against all previously chosen numbers. The first time round, K% is

restricted to the range 1 TO 1, so Card-2's number is checked only against Card-1's. If they happen to match, line 120 bounces us back to line 110 to pick a new number, and begin checking all over again. In the beginning, this program will accumulate a shuffled deck quite rapidly, but as the shuffled portion increases in size so too does the chance of choosing a number which has already been selected. And don't forget, each new random number has to be checked against an ever-increasing number of already-chosen cards. So each new random card becomes increasingly difficult to find, with the program slowing down to a snail's pace as it tries to shuffle the few remaining cards.

Yet I've seen this very routine, or variations of it, in all kinds of programs where random sequences are required over a given range, but with each number different from all others selected.

Here's a slightly better way to do the same thing :

```
100 FOR I% = 1 TO 52: B%(I%) = 1: NEXT I%
110 FOR I% = 1 TO 52
120 J% = RND(0) * 52 + 1
130 IF B%(J%) = 0 GOTO 120 ELSE B%(J%) = 0
140 C%(I%) = J%: NEXT I%
```

This runs faster because we set up an array, B%, in Line 100, and initialise it completely to 1. Then, for each card in sequence, we choose a random number, J%, from the range 1 to 52, but this time, instead of checking it against all previously chosen numbers, we look in the B%-array to see if it's already been selected. If it's OK, we set B%(J%) to 0 to indicate "This number has been chosen" and set C%(I%) to J%. And so on for the whole 52 cards. Even this program slows down drastically as the available numbers get used up, and it has to keep on trying new random numbers to find an unused one, but at least it "knows" almost instantaneously whether the new number is valid or not!

However, there's a much better way to do the job! It's **FAST** all the way through and gets the job done in a fraction of the time.

```
100 FOR I% = 1 TO 52: C%(I%) = I%: NEXT I%
110 FOR I% = 52 TO 2 STEP -1 : J% = RND(0) * I% + 1
120 SWAP C%(I%), C%(J%): NEXT I%
```

Do you see how this one works? We use only one array, C%, to hold the numbers in numerical sequence to begin with, and also to shuffle them around. Line 100 does the initialising, then in line 110 we imagine ourselves looking along the line of the array from the high end (Card 52). Next we select a random card from the deck of 52 (say Card-23) and swap it with Card-52. So now Card-23 is in 52's position and 52 is in 23's position. Then we slide down the deck by one card, ignoring the 52nd position altogether, and select a random card from the remaining 51 to swap with Card-51. Just by chance, of course, it could be position 23 again (now holding the 52 card), but more than likely it'll be some other card which has to be swapped into position 51. And so on down through the remaining deck, which is itself getting randomised along the way, until there are only two cards left. We randomly choose one of these, which, of course, only leaves the card in position-1, and as we have no choice left for this final card, we simply abandon the I%-loop at this point.

And so we now have our randomised sequence stored in the C%-array, which took only 51 times through the loop to accomplish! Whenever friend Denis and I write a card-game program (BRIDGE being our latest effort as of 31 Dec '87), we use this last method, which I hope you'll find useful in your arsenal of programming quickies! Although I've mentioned card-games in particular, it can obviously be used to produce any non-repeating sequence of randomised integers. I'll be in touch.

Dear Don,

It's surprising how things seem to acquire a momentum of their own! For instance, it's not all that long ago that I ran out of steam on XBASIC EXPLANATIONS, and now, just because one reader got me started again it seems I've just got to keep rolling, always with the hope that this is what readers want, of course. Anyway, before continuing with X-X, I'd like to contribute a little patch to FLEX itself.

Problem is that several utilities, even major ones (STYLO being a case in point) return to FLEX with the Direct-Page Register left set to other than Page-Zero. This was brought home to me some time ago when an RBASIC user mentioned that RBASIC would malfunction if it were called immediately after using STYLO. RBASIC is OK, as it doesn't make use of Page-Zero! At the time I merely amended RBASIC so that it would initialise the DP-register to 00, and didn't think too much more about it. Until one day I found that DISKEDIT bombed out under similar circumstances, sometimes merely returning me to GSKBUG, and at others bombing FLEX and producing lines looking like ignition-interference running up the screen.

I realised that "fixing" STYLO wasn't the complete answer, as there are lots of other programs that do the same thing. Besides, if an emergency situation occurred where I had to leave STYLO under abnormal conditions, I'd still have no guarantee that the DP-register was correctly restored. Equally it was totally out of the question to fix each and every program which used Page-0 (as I'd done with RBASIC), as this would involve patching lots of major programs like XPC (TSC's Extended Pre-Compiler), which tended to "screw-up" just on certain statements, such as LSET, for some strange reason. But only if called immediately after programs like STYLO.

So ... I decided the easiest way was to fix FLEX itself so that on Warm-Start it would correctly initialise the DP-register. Obviously this would have to be done somewhere between Warm-Start and the point at which it put up its +++ prompt, but there didn't seem to be any room to spare where I could squeeze in the necessary code. How what? I noticed that there were a few calls to subroutines, so maybe there was room in one of these ... maybe! And guess what? Luck was with me, because I found that the very first subroutine call, BD DEL8, (at least in GSK-FLEX9) was to a vector, 7E DE39, which in turn was a simple RTS. So that BD DEL8 did nothing more than go to an RTS and bounce right back.

Luck was more than with me! BD DEL8 took up three bytes of code and that was exactly what I needed to initialise the DP-register to 00!! And now all my programs function correctly after using STYLO, or any other program which destroys the DP-register. For what it's worth, here's the patch. Look for this code

```
CD67 10CE C07F
CDB3 BD DEL8
CD6E 8E CD03
CD71 BF CC16
```

and change the BD DEL8 to

```
4F Clear A
1F 88 Set DP-reg to 00
```

Addresses may be slightly different in your version of FLEX, and you should also trace out your BD DEL8 (or whatever) to make sure that it too boils down to a mere RTS. CD67 is my Warm-Start entry-point, but be careful, as the Cold-Start entry-point at CD57 also commences with 10CE C07F.

You'll be amazed at how many of those weird, unexplained bomb-outs just don't occur any more!

Seems like I've gone on long enough for this time, but I thought readers might like to hear the story behind these "fixes", rather than just a "Here's a patch to



FLEX. Bang! Bang! Hope you like it!" style of writing. I did have quite a lot to say on logic functions in BASIC (I'll use this term to cover both XBASIC and RBASIC, or even other general-purpose BASICs), beyond what I said earlier in X-X, but I guess I'll have to leave that till next time now. Have a good New Year everyone!!

Don Williams,  
68 Micro Journal,  
5900 Cassandra Smith Road,  
Hixson, TN 37343

Sincerely,

Bob

R. Jones  
President

PS I'm now in the middle of writing Mile 34 of "Logically Speaking", which looks like it'll finish somewhere around Mile 36 or 37. This has been a major undertaking for me, but has certainly helped hone my writing skills - not that they're sufficiently developed yet to be called "skills".



**MOTOROLA INC.**

Microprocessor Products Group  
6501 William Cannon Drive West  
Austin, Texas 78735-8598

**EDITORIAL CONTACT**  
Zachary Nelson  
Cunningham Communication, Inc.  
(408) 982-0400

**READER CONTACT**  
Dean Mosley  
Motorola Inc.  
(512) 440-2839

#### MOTOROLA RELEASES PRELIMINARY INFORMATION ON NEW RISC PROCESSOR LINE

Family Name and Limited Specifications Revealed  
17 MIPS Product Stated for Second-Quarter Unveiling

AUSTIN, Texas, Feb. 17, 1988—Motorola Inc. today released preliminary specifications of its new line of reduced instruction set computing (RISC) microprocessors. The product family, officially named the 88000 provides the highest performance available in 32-bit microprocessors.

The 88000 is a three-chip set containing a primary processor and two cache memory management units (CMMUs). The chip's performance is 17 million instructions per second (MIPS) and 34,000 Dhynstones. It generates over 50 MIPS in parallel processing designs. More than 200 companies are reviewing specifications of the processor, and early samples are being evaluated by a limited number of system vendors.

Motorola did not release the remaining details on the microprocessor's architecture and availability. The company will unveil the 88000 and provide a complete set of specifications in the second quarter of 1988.

According to Motorola, markets for the 88000 include telecommunications, artificial intelligence, graphics, 3D animation, engineering design, scientific simulation, multiuser environments, parallel processing systems and supercomputers.

#### Coexistence of 68000 and RISC

Motorola emphasized its commitment to the 68000 microprocessor family and asserted that the market will support both of the company's 32-bit microprocessor lines.

"Just as the 68000 created the workstation market, the 88000 will open new markets that do not exist today," said Goldman. "While the 88000 will expand the application base by making high-performance systems such as supercomputers more affordable, the 68000 family will continue to offer the most cost-effective 32-bit performance."

Motorola's \$2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America with a balanced product portfolio of over 50,000 devices.

#### First Complete Solution

The primary processor of the 88000 contains both an integer and a floating point unit, making it the first to place these two capabilities onto a single chip. The integration of both functions provides a balanced architecture and a significant boost in performance. Coupled with the processor are two cache chips, one for data and the other for instructions (see attached diagram). This dual CMMU structure creates an efficient parallel flow of information, known as Harvard-style architecture. The CMMUs also release the system designer from the expensive and time-consuming task of implementing a separate cache-memory scheme.

The 88000 is initially being manufactured in 1.5 micron HCMOS, a high-speed derivation of CMOS (complementary metal oxide semiconductor) technology. According to Dataquest, Motorola makes more than one-third of the world's CMOS products (\$457 million in 1986). Ultimately, the new product line will move to sub-micron technology as well as ECL (emitter coupled logic), a process used to manufacture high-performance semiconductors.

"Manufacturing excellence is crucial to delivering and enhancing 32-bit microprocessors," said Murray Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group in Austin, Texas. "By maintaining control of the manufacturing process, we guarantee the quality that the industry depends on."

#### Scoreboarding Speeds Software Development

Motorola reported today that the 88000 uses an advanced technique called scoreboarding that simplifies the design of compilers and applications software. The scoreboard allocates the primary processor's register set to accommodate the concurrent execution and manipulation of multiple instructions. This allows software writers to utilize the complete register set without defining the exact progression of information as it makes its way through the chip, thus reducing the time required to write optimized RISC software.

**EDITORIAL CONTACT:**  
Bob King  
512/928-6141

**READER CONTACT:**  
Dean Mosley  
512/440-2839

**INQUIRY RESPONSE:**  
Technical Info Center  
P.O. Box 52073  
Phoenix, AZ 85072

#### MOTOROLA ANNOUNCES THE AVAILABILITY OF THE MC88030 USER'S MANUAL TO SUPPORT DESIGNERS OF THEIR SECOND GENERATION ENHANCED 32-BIT MICROPROCESSOR

Austin, Texas, January 29, 1988—Motorola's Microprocessor Products Division announces the immediate availability of the user's manual for the MC88030, (030). The MC88030 User's Manual contains descriptions of the microprocessor's advanced capabilities, operations and programming details.

The 030 (oh thirty) is the newest member of the 68000 software-compatible microprocessor family. The oh thirty provides up to twice the performance of the most powerful and widely used 32-bit microprocessor, Motorola's MC68020 (020). The 030 is the first microprocessor to have on-chip data and instruction caches, parallel (Harvard-style) architecture, and both synchronous and asynchronous bus interfaces, and includes a memory management unit.

Within the User's Manual there are 14 different sections plus an appendix which summarizes the 68000 Family.

- Section 1. Vocabulary Information
- Section 2. Data Organization & Detailed Addressing Capabilities
- Section 3. The Instruction Set
- Section 4. Processing States
- Section 5. Signal Descriptions
- Section 6. On-Chip Cache Memories
- Section 7. Bus Operation
- Section 8. Exception Processing
- Section 9. Memory Management Unit
- Section 10. Coprocessor Interface Description
- Section 11. Instruction Execution Timing
- Section 12. Applications Information
- Section 13. Electrical Specifications
- Section 14. Ordering Information and Mechanical Data

Motorola Inc.  
Literature Distribution Center  
P.O. Box 20924  
Phoenix, AZ 85036-0924



**MOTOROLA INC.**

**Microprocessor Products Group**  
**8501 William Cannon Drive West**  
**Austin, Texas 78735-8596**

**EDITORIAL CONTACT:**  
 Bob King  
 512/928-8141

**READER CONTACT:**  
 Rhonda Alexis Drvin  
 512/440-2140

**INQUIRY RESPONSE:**  
 Technical Info Center  
 P.O. Box 52073  
 Phoenix, AZ 85072

**MOTOROLA INTRODUCES ENHANCED TOKEN BUS CONTROLLER FOR REAL-TIME COMMUNICATION NETWORKS**

*Price cut 43% on MC68824 Token Bus Controller, speed pushed to 16MHz, and now offered in a new low cost Plastic Leaded Chip Carrier (PLCC) package*

Austin, Texas, January 20, 1988.....The Motorola Microprocessor Products Division is introducing a new Enhanced Token Bus Controller chip (Enhanced TBC) designed for cost sensitive, high performance real-time communication networks. Such networks for factory communications are specified by the Enhanced Performance Architecture (EPA) portion of Manufacturing Automation Protocol (MAP) 3.0. By performing IEEE 802.2 Logical Link Control (LLC) type 3 acknowledged connectionless services on-chip, the Enhanced TBC significantly increases the real time performance of the network and significantly decreases the amount of software the programmer is required to write. The Enhanced TBC is price competitive with other network solutions such as CSMA/CD (Ethernet or IEEE 802.3) while offering 1st greater reliability and capability.

According to Rhonda Alexis Drvin, Motorola LAN VLSI Program Manager, "The present MC68824 Token Bus Controller has been designed into high performance, high speed applications over the past year. The new Enhanced TBC opens the doors to both lower speed and real time performance networks in addition to the high performance applications it already supports." These economical markets include reconstruction of sensors, low cost, low level factory controllers, programmable controllers, and environmental building control that will benefit from the robustness of the IEEE 802.4 protocol."

The Enhanced TBC is specified to operate at low serial data rates down to 10kbps - ideal for low cost applications such as Instrument Standards Association (ISA) SP-50 field bus, currently not addressed by MAP. In these applications, physical connection is avoided so the connection to MAP is simple and inexpensive. The Enhanced TBC can also be used for PBX backbone local interconnections.

Other improvements, such as faster IEEE 802.4 protocol implementation, bridging options and increased network reliability, have been implemented in the Enhanced TBC. The Enhanced Token Bus Controller is 100% upward compatible with Motorola's existing Token Bus Controller widely used today worldwide. Performing IEEE 802.2 Logical Link Control (LLC) type 3 acknowledged connectionless services on-chip, the Enhanced TBC dramatically increases the real time performance of the network and significantly decreases the required supporting software.

The speed of the TBC has been pushed up to 16MHz in addition to already available 10 and 12MHz versions. The new Enhanced TBC is available at 10, 12.5 and 16MHz speeds. The Enhanced Token Bus Controller is currently available in an 84-lead pin grid array package. A PLCC package version will be available in 1Q88. Pricing for 100 piece quantities of both the current TBC (MC68824RC10) and the Enhanced TBC (MC68824RC10E) is \$43.30 in the PLCC package. For more information, please contact your local Motorola sales representative.

**EDITORIAL CONTACT:**  
 Diane Falkenberg  
 512/928-8869

**READER CONTACT:**  
 Laura Tolpin  
 512/928-2035

**MOTOROLA ISSUES APPLICATION NOTE FOR SERIAL PERIPHERAL INTERFACE**

The application note describes conditions under which SPI may provide a solution to problems encountered in such communication, especially when differing processors are used.

SPI systems can be operated in master or slave modes; the slave being useful when the MCU is selected for use as a smart peripheral or as a slave processor. The SPI subsystem is also effective in multi-master systems. It is compatible with many of the industry's serial communication peripheral IC's.

The SPI is especially useful as a master when additional off-chip peripheral circuits are required in the system. Motorola's SPI compatible peripherals include:

MC145041	8-bit A/D Converter
MC14499	7-Segment LED Display Decoder/Driver
MC14500	LCD Driver
MC144110	D/A Converter
*MC145051	10-bit A/D Converter

The new application note also describes a special case which occurs when one or more of the microcontrollers in a circuit do not have SPI capability in hardware. A simple software routine can be written to perform the interface in such cases, avoiding the need for external hardware alternatives and simplifying design efforts.

For a copy of the application note (AN991/D), contact the Motorola Literature Distribution Center, P. O. Box 20912, Phoenix, AZ 85 36, your local Motorola Sales Office or Motorola Distributor.

**GESPAC Inc.**  
 50 West Hoover Ave.  
 Mesa, Arizona 85210  
 Tel. (602) 962-5559  
 Fax. (602) 962-5750

**Reader Contact:** Mark Stephens  
**Editorial Contact:** Cosma Pabouctsidis

**GESPAC SLASHES PRICES OF 68020 BOARDS**

Mesa, AZ., March 4, 1988--GESPAC has dramatically reduced the price of the GESMPU-20, 68020 CPU board for the G-64 bus. At \$995 for single quantity orders, the GESMPU-20 is now the least expensive implementation of 68020 CPU on standard board level products.

The GESMPU-20 features a 12.5 MHz 6800 CPU, 256K of zero-wait-states CMOS RAM, sockets for up to 512 Kibytes of EPROM and socket for the 68881 arithmetic coprocessor. The board also exists equipped with 512 Kibytes of RAM and a 16.7 MHz clock.

The board is fully expandable to use the selection of over 150 board level functions offered by GESPAC on the G-64 bus. The OS-9 real-time, multi-tasking operating system is available for the GESMPU-20.

**GESPAC INTRODUCES VERSATILE 6809 SINGLE BOARD COMPUTER**

MESA, AZ, January 22, 1988--GESPAC introduces a new single board microcomputer featuring the 6809 microprocessor aimed at imbedded instrument and machine control applications. The GES3B5-4A fits on a single height Eurocard and is compatible with the standard G-64 bus.

The GES3B5-4A is equipped with 3 JEDEC sockets that can accommodate up to 32 Kbytes of EPROM and up to 16 Kbytes of CMOS RAM. The RAM area is powered out of an on-board lithium battery that switches in automatically in the event of a power failure and prevents data loss for over two years.

The board provides the user with 40 TTL parallel I/O lines, one RS-232 serial port, four 16-bit timers, and a real time clock/calendar powered by the on-board battery. The GK5883-4A also is also equipped with a "watchdog" timer which protects the module against loss of program control.

The board can be used in stand alone operation or in conjunction with any of the over 150 I/O and memory modules offered by GE5PAC for the G-64 bus.

The G-64 bus is an easy-to-interface second generation 16-bit bus aimed at midrange industrial applications.

The GK5883-4A is supported by GE5PAC with the OS-9 real-time multitasking operating system.

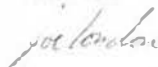
The GK5883-4A is available today at the low unit price of \$495.

Joseph D. Condon  
8072 132nd St W.  
Lehewille, MN 55044  
Phone 612-431-7624

Dear Sir,

I would like to bring to your attention a typographical error that occurred in my article "Back on a Budget" as it appeared in the January issue of 68MJ. On page 48, listing #1, line 180 should read '17 W(1) > H W(1)'. I am sorry for any inconvenience this might have caused you or your readers.

Sincerely,



Joseph D. Condon



#### BOOKMARK MACINTOSH DEBUTS

Novato, CA January 18 - IntelliSoft has released a new version of BOOKMARK disaster recovery software to support Apple Computer's Macintosh Plus and SE Models. This new product, which marks the third by IntelliSoft International, represents an unprecedented breakthrough in the microcomputer marketplace since it is the first software to back up RAM, or system memory.

"We're excited to be in the Macintosh arena- we think it has a very bright future," said company president Bruce Bierman. He added that the BOOKMARK concept is a unique one because it places the burden of backup on the computer, not the user, who is then free to maximize time on more creative tasks.

BOOKMARK is a software disk accessory designed to provide fault tolerant computing by automatically storing unsaved work to the hard disk. This "memory backup" prevents loss of data due to power failure, accidental power down or crash, system errors (the "bomb"), or frozen keyboard. After reboot, recovery takes just seconds and requires only a single keystroke or mouse click.

Another option allows Macintosh systems without hard disks to utilize BOOKMARK. This feature invokes automatic file saves to floppy disk at user definable intervals.

Company:  
MaryAnn Phillips  
IntelliSoft International  
51 Digital Drive  
Novato, CA 94949  
Tel (415) 883-1188  
FAX (415) 883-2646  
Telex 470766

## Classifieds As Submitted - No Guarantees

MUSTANG-020 16Mhz with 68881, OS9 Professional Package & C \$3000.  
Call Tom (615) 842-4600.

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

#### DAISY WHEEL PRINTERS

Qume Sprint 9 - \$900

Qume Sprint 5 - \$800.

HARD DISK 10 Megabyte Drive - Seagate Model #412 \$275.  
3-Dual 8" drive enclosure with power supply. New in box. \$125 each.  
5-Siemens 8" Disk Drive, \$100 each.  
Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, \$495.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09CPU Card- \$900 complete.

Tom (615) 842-4600 M-F 9AM to 5PM EST

...

GMX MICRO-20 (16.67 MHz) with MC68881 and OS9/68020 professional pak \$3500 or best offer.

John Bing 9-5 EST (301) 428-8214

...

Complete Set of 68 Micro Journals from February 1979, volume 1, issue 1, thru current. All in good or excellent condition. Asking \$250.

Also, two complete, 6 year old, SWTPC 40 Meg, CDS-2 hard disk drives, cabinets, cables, documentation and controllers. Power supplies & controller cards work but drives have "cannot open or read device" errors on trying to mount. Will consider any offer of purchase on "as is" basis.

Dwight Lanpher, Box 472, NE Harbor, ME 04662, (207) 276-5350



# APPLE

# MACINTOSH™



# USERS

**Save over a \$1,000.00**  
**on PostScript**  
**Laser Printers!**  
**Faster - Finer Quality**  
**than the original Apple**  
**LaserWriter!**  
**New & Demos**  
**Cartridges-new-rebuilds**  
**-colors-**

**In Chattanooga Call:**  
**615 842-4600**  
**QMS-Authorized**

#### Data-Comp Division

A Decade of Quality Service™  
Systems World Wide  
Computer Publishing, Inc. 5805 Cassandra Smith Road  
Telephone 615-842-4601 Telex 510 800-6600 Hixson, TN 37343

## SUPERIOR SOFTWARE FOR YOUR 6809

★ SK-OOS® Disk Operating System .....	\$75.00
★ Configuration Manual .....	\$50.00
★ HUMBUG® Monitor .....	\$50.00
★ MICROBUG Monitor .....	\$30.00
★ SPELL 'N FIX Spelling Checker .....	\$89.29
★ STAR-DOS for the Coco .....	\$34.50
★ CHECK 'N TAX .....	\$50.00

## AND 68000

★ SK-DOS® Operating System .....	\$140.00
★ HUMBUG® Monitor .....	\$ 50.00
★ SPELL 'N FIX Spelling Checker (Coming)	



SOFTWARE SYSTEMS CORPORATION  
BOX 209 • MT. KISCO, N.Y. 10549 • 914/241-0287

## OMEGASOFT

### 6809 PASCAL CLOSE-OUT SALE 50% OFF ALL 6809 HOST PRODUCTS

In March and April you can purchase OmegaSoft Pascal 6809 host products at 50% off our regular price, direct from Certified Software Corp., or through participating dealers. The following products are available:

**PCS2 + RALL1**: Includes compiler, assembler, linker, debugger with source code, and runtime library with source code. \$275  
**SEK1**: Screen Editor Kit. Configurable for various terminals. \$45

**APU1**: Allows use of AMD9511 chip for integer, long integer, and real arithmetic. \$45

**MTK1**: Multi-tasking kernel. Allows task procedures without an operating system in your target system. \$85

Available for OS-9 (Microware), FLEX (TSC), MDOS and XDOS (Motorola) on 5" or 8" SSD format.

Shipping charges extra. Mastercard and Visa accepted.

These products to be discontinued after April.

OmegaSoft is a registered trademark of Certified Software Corporation.

**CERTIFIED  
SOFTWARE  
CORPORATION**

616 CAMINO CABALLO, NIPOMO, CA 93444  
TEL.: (805) 929-1395 TELEEX; 467013  
FAX; (805) 929-1395 (MID-8AM)

## SOFTWARE FOR 680x AND MSDOS

### SUPER SLEUTH DISASSEMBLERS

**EACH \$99-FLEX \$101-OS9 \$100-UNIFLEX**  
**OBJECT-ONLY versions: EACH \$50-FLEX, OS9, COCO**  
Interactively generate source on disk with labels, include xref, binary editing  
specify 6800, 1.2, 3.5, 6.8/6502 version or Z80/8080, 5 version  
OS9 version also processes FLEX format object file under OS9  
COCO DOS available in 6800, 1.2, 3.5, 6.8/6502 version (not Z80/8080.5) only  
68010 disassembler \$100-FLEX, OS9, UNIFLEX, MSDOS, UNIX, SKDOS

### CROSS-ASSEMBLERS WITH MACRO CAPABILITIES

**EACH \$50-FLEX, OS9, UNIFLEX, MSDOS, UNIX, SKDOS 3/\$100 ALL/\$200**  
specify: 180x, 6502, 6801/11, 6804, 6805, 6809, Z8, Z80, 8048, 8051, 8085, 68010, 32000  
modular cross-assemblers in C, with load/unload utilities  
sources for additional \$60 each, \$100 for 3, \$300 for all

### DEBUGGING SIMULATORS FOR POPULAR 8-BIT MICROPROCESSORS

**EACH \$75-FLEX \$100-OS9 \$80-UNIFLEX**  
**OBJECT-ONLY versions: EACH \$50-COCO FLEX, COCO OS9**  
Interactively simulate processors, include disassembly formatting, binary editing  
specify for 6800/1, (14)6805, 6502, 6809 OS9, Z80 FLEX

### ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809

6502 to 6809 \$75-FLEX \$85-OS9 \$60-UNIFLEX  
6800/1 to 6809 & 6809 to position-ind. \$50-FLEX \$75-OS9 \$60-UNIFLEX

### FULL-SCREEN XBASIC PROGRAMS with cursor control

**AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS**  
DISPLAY GENERATOR/DOCUMENTOR \$50 w/source, \$25 without  
MAILING LIST SYSTEM \$100 w/source, \$50 without  
INVENTORY WITH MRP \$100 w/source, \$50 without  
TABULA RASA SPREADSHEET \$100 w/source, \$50 without

### DISK AND XBASIC UTILITY PROGRAM LIBRARY

**\$50-FLEX \$30-UNIFLEX/MSDOS**  
edit disk sectors, sort directory, maintain master catalog, do disk sorts,  
resequence some or all of BASIC program, xref BASIC program, etc.  
non-FLEX versions include sort and resequencer only

### MODEM TELECOMMUNICATIONS PROGRAM

**\$100-FLEX, OS9, UNIFLEX, MS-OOS, UNIX, SKDOS**  
**OBJECT-ONLY versions: EACH \$50**  
menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.  
for COCO and non-COCO, drives internal COCO modem port up to 2400 Baud

## DISKETTES & SERVICES

### 5.25" DISKETTES

**EACH 10-PACK \$7.50-SSSD/SSDD/DSDD**

American-made, guaranteed 100% quality, with Tyvek jackets, hub rings, and labels

### ADDITIONAL SERVICES FOR THE COMPUTING COMMUNITY CUSTOMIZED PROGRAMMING

We will customize any of the programs described in this advertisement or in our  
brochure for specialized customer use or to cover new processors; the charge  
for such customization depends upon the maintainability of the modifications.

### CONTRACT PROGRAMMING

We will create new programs or modify existing programs on a contract basis.  
a service we have provided for over twenty years; the computers on which we  
have performed contract programming include most popular models of  
mainframes, including IBM, Burroughs, Univac, Honeywell, and most popular  
models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most  
popular brands of microcomputers, including 6800/1, 6809, Z80, 6502,  
680x0, using most appropriate languages and operating systems, on systems  
ranging in size from large telecommunications to single board controllers;  
the charge for contract programming is usually by the hour or by the task.

### CONSULTING

We offer a wide range of business and technical consulting services, including  
seminars, advice, training, and design, on any topic related to computers;  
the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants, Inc.  
1454 Lettie Lane, Conyers, GA 30207  
Telephone 404-463-4570 or 1717

We take orders at any time, but plan  
long discussions after 5, if possible.

Contact us about catalog, dealer, discounts, and services.  
Most programs in source: give computer, OS, disk size.  
25% off multiple purchases of same program on one order.  
VISA and MASTER CARD accepted; US funds only, please.  
Add GA sales tax (if in GA) and 5% shipping.

(UNIFLEX in Technical Systems Consultants; OS/9 Microware;  
COCO Tandy; MSDOS Microsoft; SKDOS Stark Software)

# K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9  
FLEX or SK\*DOS  
Even runs on the 68XXX SK\*DOS Systems\*

*Hundreds Sold at  
Suggested Retail:*

~~\$199.00~~

• 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary ".CMD" program. Much faster than BASIC programs.

• 6809 - FLEX users can compile their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

• 68XXX - SK\*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK\*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up in the "Learn a New Language" syndrome - Write Your Program in BASIC, Debug It In BASIC and Then Compile It to a .CMD Binary File.

For a LIMITED time  
save over 65%...  
This sale will not be  
repeated after it's  
over! \*

SALE SPECIAL:

**\$69.95**

## SPECIAL Thank-You-Sale

Only From:

**CPI**

**S.E. Media™**

5900 Cassandra Smith Rd.  
Hixson, Tn 37343  
Telephone 615 842-6809  
Telex 510 600-6630

A Division of Computer Publishing Inc.  
Over 1,200 Titles - 6800-6809-68000

\* K-BASIC will run under 68XXX SK\*DOS in emulation mode for the 6809.

Price subject to change without notice.

# Clearbrook Software Group

(604)853-9118



CSG IMS is *THE* full featured relational database manager for OS9/OSK. The comprehensive structured application language and B + Tree index structures make CSG IMS the ideal tool for file-intensive applications.

CSG IMS for CoCo2/3 OS9 L1/2 (single user)	\$169.95
CSG IMS for OS9 L2 or 68000(multi user)	\$495.00
CSG IMS demo with manual	\$30

**MSF - MSDos File Manager for CoCo 3/OS9 Level 2**  
allows you to use MSDos disks directly under OS9.  
Requires CoCo 3, OS9 L2, SDISK3 driver \$45.00

## SERINA - System Mode Debugger for OS9 L2

allows you to trace execution of any system module, set break points, assemble and disassemble code and examine and change memory.

Requires CoCo3 or Gimix II, OS9 L2 & 80 col. terminal \$139.00

## ERINA - Symbolic User Mode Debugger for OS9

lets you find bugs by displaying the machine state and instructions being executed. Set break points, change memory, assemble and disassemble code.

Requires 80 column display, OS9 L1/2 \$69.00

Shipping: N. America - \$5, Overseas - \$10  
Clearbrook Software Group P.O. Box 8000-499, Sumas, WA 98295  
OS9 is a trademark of Microware Systems Corp., MSDos is a trademark of Microsoft Corp.

# SPECIAL

## ATARI™

### &

## OS-9™

**NOW!**

If you have either the  
Atari 520 or 1040 -  
you can take  
advantage of the  
"bargain of a lifetime"  
OS-9 68K and BASIC  
all for the low, low price of:

# \$150.00

Call or Write

S.E. Media

5900 Cassandra Smith Rd.

Hixson, TN 37343

615 842-4601

## ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage for the **Atari & Amiga™** series of 68000 computers.

Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari & Amiga computers. Especially since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

**This I must stress - Input from you on the Atari & Amiga. As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!**

DMW



# THE 6800-6809 BOOKS

..HEAR YE.....HEAR

## OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibble's **OS9 USER NOTES**

Information for the BEGINNER to the PRO,  
Regular or CoCo OS9

### Using OS9

HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS,  
OS9 STANDARDS, Generating a New Bootstrap, Building a  
new System Disk, OS9 Users Group, etc.

### Program interfacing to OS9

DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION,  
"SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

### Programming Languages

Assembly Language Programs and Interfacing; Basic09, C,  
Pascal, and Cobol reviews, programs, and uses; etc.

### Disks Include

No typing all the Source Listings in. Source Code and,  
where applicable, assembled or compiled Operating  
Programs. The Source and the Discussions in the  
Columns can be used "as is", or as a "Starting Point"  
for developing your OWN more powerful Programs.  
Programs sometimes use multiple Languages such as a  
short Assembly Language Routine for reading a  
Directory, which is then "piped" to a Basic09 Routine  
for output formatting, etc.

## BOOK \$9.95

Typeset -- w/ Source Listings  
(3-Hole Punched: 8 x 11)

Deluxe Binder - - - - - \$5.50

### All Source Listings on Disk

1-8" SS, SD Disk - - - \$14.95

2-5" SS, DD Disk - - - \$24.95

Shipping & Handling \$3.50 per Book, \$2.50 per Disk set

Foreign Orders Add \$4.50 Surface Mail  
or \$7.00 Air Mail

If paying by check - Please allow 4-6 weeks delivery

\* All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.  
5900 Cassandra Smith Rd.  
Hixson, TN 37343



\*FLEX is a trademark of Technical Systems Consultants  
\*OS9 is a trademark of Microware and Motorola  
\*68' Micro Journal is a trademark of Computer Publishing Inc.

## FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGOC1	File load program to offset memory — ASM PIC
MEMOVE C1	Memory move program — ASM PIC
DUMP C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT C3	Parallel (enhanced) printer driver — ASM
MODEM C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG C1	Scientific math routines — PASCAL
UC4	Mini-monitor, disk resident, many useful functions — ASM
PRINT C4	Parallel printer driver, without PFLAG — ASM
SET C5	Set printer modes — ASM
SETBAS1 C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

\*\*Over 30 TEXT files included is ASM (assembler)-PASCAL-  
PIC (position independent code) TSC BASIC-C, etc.

Book only: \$7.95 + \$2.50 S/H

With disk: 5" \$20.90 + \$2.50 S/H

With disk: 8" \$22.90 + \$2.50 S/H



(615) 842-4601  
Telex 5106006630

# !!! Subscribe Now !!! 68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard ☐ VISA ☐

Card # \_\_\_\_\_ Exp. Date \_\_\_\_\_

For 1 Year \_\_\_\_ 2 Years \_\_\_\_ 3 Years \_\_\_\_

Enclosed: \$ \_\_\_\_\_

Name \_\_\_\_\_

Street \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Country \_\_\_\_\_

My Computer Is: \_\_\_\_\_

## Subscription Rates

U.S.A.: 1 Year \$24.50, 2 Years \$42.50, 3 Years \$64.50

\*Foreign Surface: Add \$12.00 per Year to USA Price.

\*Foreign Airmail: Add \$48.00 per Year to USA Price.

\*Canada & Mexico: Add \$9.50 per Year to USA Price.

\*U.S. Currency Cash or Check Drawn on a USA Bank !

**68 Micro Journal**  
5900 Cassandra Smith Rd.



POB 849  
Hixson, TN 37343



Telephone 615 842-4600  
Telex 510 600-6630

## Reader Service Disks

- Disk-1 Filesort, Minicat, Minicopy, Minifms, \*\*Lifetime, \*\*Poetry, \*\*Foodlist, \*\*Diet.
- Disk-2 Diskedit w/ inst. & fixes, Prime, \*Pmmod, \*\*Snoopy, \*\*Football, \*\*Hexapawn, \*\*Lifetime.
- Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, \*Disksave.
- Disk-4 Mailing Program, \*Findat, \*Change, \*Testdisk.
- Disk-5 \*DISKFIX 1, \*DISKFIX 2, \*\*LETTER, \*\*LOVESIGN, \*\*BLACKJAK, \*\*BOWLING.
- Disk-6 \*\*Purchase Order, Index (Disk file indx).
- Disk-7 Linking Loader, Rload, Harkness.
- Disk-8 Crtesi, Lanpher (May 82).
- Disk-9 Datecopy, Diskfix9 (Aug 82).
- Disk-10 Home Accounting (July 82).
- Disk-11 Dissembler (June 84).
- Disk-12 Modem68 (May 84).
- Disk-13 \*Initmf68, \*Termf68, \*Cleanup, \*Dskalign, Help, Date, Txt.
- Disk-14 \*Init, \*Test, \*Terminal, \*Find, \*Diskedit, Init.Lib.
- Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
- Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
- Disk-17 Match Utility, RATBAS, A Basic Preprocessor.
- Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray).
- Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
- Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist).
- Disk-21 Dragon.C, Grep.C, L.S.C, FDUMP.C.
- Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
- Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.
- Disk-23 ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March 1986.
- Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
- Disk-25 KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
- Disk-26 Compacta UniBoard review, code & diagram, Burlison March '86.
- Disk-27 ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
- Disk-28 CT-82 Emulator, bit mapped.
- Disk-29 \*\*Star Trek
- Disk-30 Simple Winchester, Dec. '86 Green.
- Disk-31 \*\*\* Read/Write MS/PC-DOS (SK \*DOS)
- Disk-32 Heir-UNIX Type upgrade - 68MJ 2/87
- Disk-33 Build the GT-4 Terminal - 68MJ 11/87 Condon.
- Disk-34 FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - 68MJ 4/88 Korpi.

### NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

\* Denotes 6800 - \*\* Denotes BASIC

\*\*\* Denotes 68000 - 6809 no indicator.



8" disk \$19.50  
5" disk \$16.95



Shipping & Handling -U.S.A. Add: - \$3.50  
Overseas add: \$4.50 Surface - \$7.00 Airmail

## 68 MICRO JOURNAL

5900 Cassandra Smith Rd.  
Hixson, TN 37343  
(615) 842-4600 - Telex 510 600-6630

# PT-68000 SINGLE BOARD COMPUTER

The PT68K2 is Available in a Variety of Formats  
From Basic Kits to Completely Assembled Systems

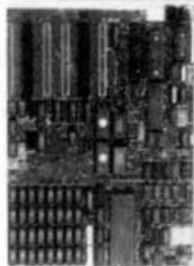
**BASIC KIT (8 MHZ)** - Board, 68000,  
HUMBUG MONITOR + BASIC in ROM,  
4K STATIC RAM, 2 SERIAL PORTS, all  
Components \$200

**PACKAGE DEAL** - Complete Kit with  
Board 68000 10 MHZ, SK'DOS, 512K  
RAM, and all Necessary Parts \$530

**ASSEMBLED BOARD (12 MHZ)**  
Completely Tested, 1024K RAM,  
FLOPPY CONTROLLER, PIA, SK'DOS  
\$849

**ASSEMBLED SYSTEM** - 10 MHZ  
BOARD, CABINET POWER SUPPLY,  
MONITOR + KEYBOARD, 80 TRACK  
FLOPPY DRIVE, CABLES \$1299  
For A 20 MEG DRIVE, CONTROLLER  
and CABLES Add \$345

**PROFESSIONAL OS9** \$500



## FEATURES

- MC68000 Processor, 8 MHZ Clock (optional 10, 12.5 MHZ)
- 512K or 1024K of DRAM (no wait states)
- 4K of SPAM (6116)
- 32K, 64K or 128K of EPROM
- Four RS-232 Serial Ports
- Floppy disk controller will control up to four 5 1/4", 40 or 80 track.
- Clock with on-board battery.
- 2 - 8 bit Parallel Ports
- Board can be mounted in an IBM type PC/XT cabinet and has a power connector to match the IBM type power supply.
- Expansion ports - 6 IBM PC/XT compatible I/O ports. The HUMBUG monitor supports monochrome and/or color adaptor cards and Western Digital winchester interface cards.

## PERIPHERAL TECHNOLOGY

1480 Terrell Mill Rd., Suite 870

Marietta, Georgia 30067

404/984-0742

VISA/MASTERCARD/CHECK/C.O.D.

Send For Catalogue

For Complete Information On All Products

\*SK'DOS is a Trademark of  
STAR-K SOFTWARE SYSTEMS CORP.  
\*OS9 is a Trademark of Microware

## DATA-COMP

## SPECIAL

### Heavy Duty Power Supplies



For A limited time our HEAVY DUTY SWITCHING POWER SUPPLY. These are BRAND NEW units. Note that these prices are less than 1/4 the normal price for these high quality units.

**Make: Boschert**

Size: 10.5 x 5 x 2.5 inches

Including heavy mounting bracket and heatsink

Rating: in 110/220 volts ac (strap change) Out: 130 watts

Output: +5v - 10 amps

+12v - 4.0 amps

+12v - 2.0 amps

-12v - 0.5 amps

Mating Connector: Terminal strip

Load Reaction: Automatic short circuit recovery

**SPECIAL: \$59.95 each**

2 or more \$49.95 each

Add: \$7.50 each SH

**Make: Boschert**

Size: 10.75 x 6.2 x 2.25 inches

Rating: 110/220 ac (strap change) Out: 81 watts

Outputs: +5v - 8.0 amps

+12v - 2.4 amps

+12v - 2.4 amps

+12v - 2.1 amps

-12v - 0.4 amps

Mating Connectors: Molex

Load Reaction: Automatic short circuit recovery

**SPECIAL: \$49.95 each**

2 or more \$39.95 each

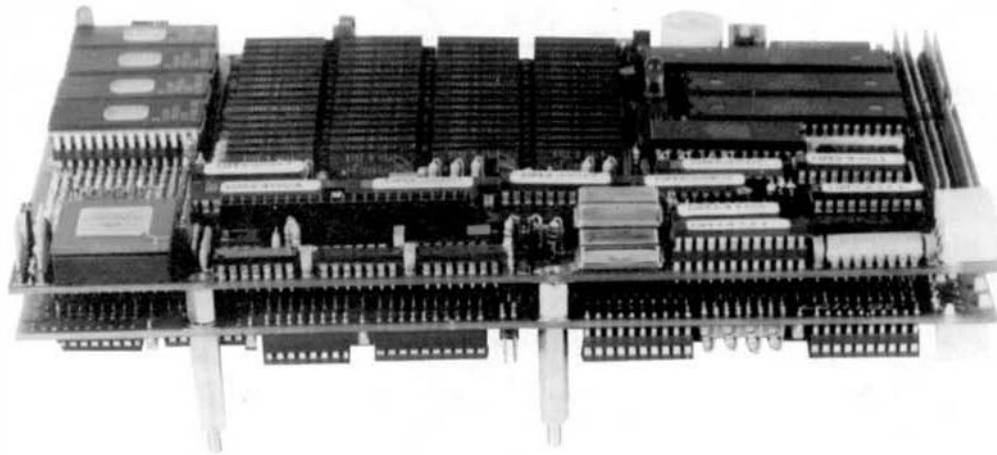
Add: \$7.50 SH each

5800 Cassandre Smith Rd., Hickory, TN. 37343

Telephone 615 842-4600

Telex 510 600-653D

# NOW THE GMX MICRO-20 HAS A TWIN — *Announcing The* **GMX TWINGLE-20** **68020 TWIN BOARD COMPUTER WITH MMU**



All the features of the GMX Micro-20, PLUS —

- 2 Megabytes additional RAM — for a total of 4 Megabytes of RAM
- 8 more Serial ports — for a total of 12, and expandable up to 44.
- MEMORY MANAGEMENT UNIT

The **GMX TWINGLE-20** consists of 2 boards. One of the boards is the same as the Micro-20, except for the 68020 processor which is on the MMU board. It uses the same I/O expansion interface, serial adapter boards, and mounting holes as the GMX Micro-20, making it easy to upgrade existing systems. Any of the currently available GMX Micro-20 I/O expansion boards can be used to provide additional I/O capability. Expansion possibilities include additional serial ports (up to 44 ports total), additional parallel ports, and local area networking of up to 255 GMX Micro-20s and/or TWINGLE-20s.

The **MMU board** contains the additional 2 Megabytes of RAM, 8 serial ports with 2 connectors for the SAB 4 port adaptor cards, and the MMU hardware. The MMU is a proprietary high-speed design that fully supports virtual memory. The system RAM normally operates with only 1 wait-state, regardless of processor speed. An additional wait-state is needed only when program flow crosses a 4K boundary. The MMU can be configured for any one of four different maps, ranging from 8 tasks with 8 megabytes of virtual address space each, to 64 tasks of 1 megabyte each. The MMU can be disabled for applications that do not use hardware memory management.

The **TWINGLE-20** two board set can occupy the same space as a half-height 5.25" disk drive. It is available in 12.5, 16.67 or 20 MHz. versions, and with or without the 68881 FPC.

## SPECIFICATIONS

Size: 8.8 x 5.75 x 1.4 inches.

Power Requirements: +5VDC @ 8.3A typical (20MHz. with 68881).

The TWINGLE-20 itself does not require a +12V supply. +12V supply requirements, if any, are determined by the serial adapter boards and any I/O expansion boards powered through the I/O Expansion Interface.

## SOFTWARE INCLUDED:

An enhanced version of D20Bug with diagnostics for the MMU and the additional RAM and serial ports.

## OPTIONAL SOFTWARE:

**UniFLEX VM**, Virtual Memory version of the UniFLEX operating system which includes all of the features of the GMX Micro-20 version, plus full MMU support.

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

- Compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 8 Megabytes of Virtual Memory per user.

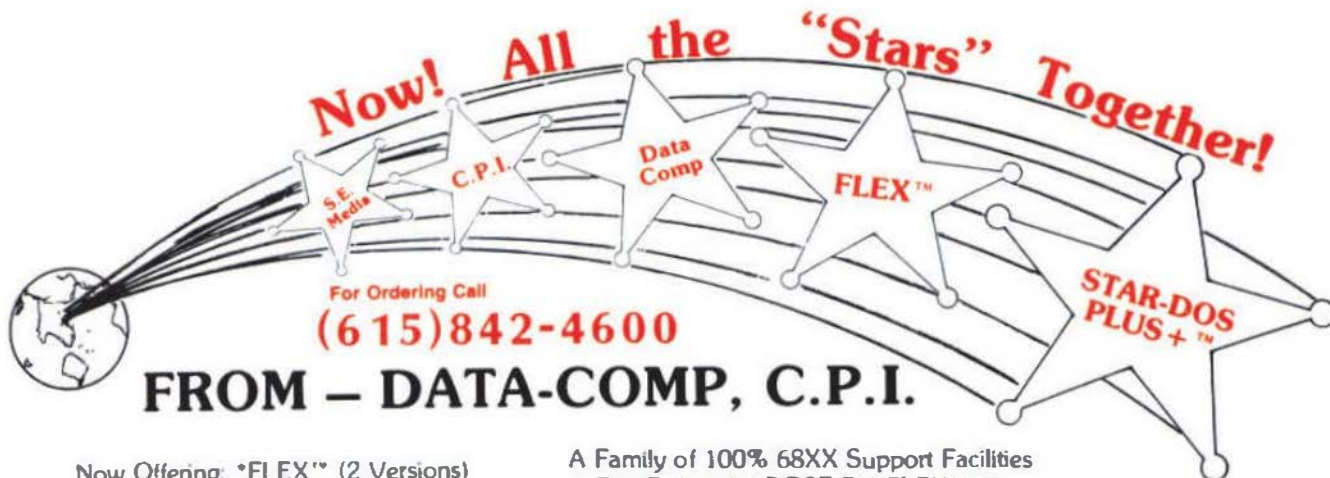
All the optional languages and software that run under UniFLEX for the Micro-20 are also available for the TWINGLE-20.

**OS-9** Users can take advantage of the additional RAM and serial ports on the TWINGLE-20. It does not presently support the MMU.

OS-9 is a trademark of Microware Systems Corp.  
UNIX is a trademark of A.T&T.

UniFLEX trademark of Technical Systems Consultants, Inc.  
GMX, GMMX and TWINGLE are trademarks of GMX Inc.

**GMX<sup>INC</sup>** 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352



Now Offering: \*FLEX™ (2 Versions)  
AND \*STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities  
The Folks who FIRST Put FLEX™ on  
The CoCo

**FLEX-CoCo Sr.**  
with TSC Editor  
TSC Assembler

Complete with Manuals  
Reg. \$250.<sup>00</sup> **Only \$79.<sup>00</sup>**

**STAR-DOS PLUS+**

- Functions Same as FLEX
- Reads - writes FLEX Disks **\$34.<sup>00</sup>**
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

**FLEX-CoCo Jr.**  
without TSC  
Editor & Assembler  
**\$49.<sup>00</sup>**

**PLUS**

**ALL VERSIONS OF FLEX & STAR-DOS INCLUDE**

**TSC Editor**  
Reg \$50.00  
**NOW \$35.00**

- + Read-Write-Dir RS Disk
- + Run RS Basic from Both
- + More Free Utilities

- + External Terminal Program
- + Test Disk Program
- + Disk Examine & Repair Program
- + Memory Examine Program
- + Many Many More!!!

**TSC Assembler**  
Reg \$50.00  
**NOW \$35.00**

**CoCo Disk Drive Systems**

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES  
SYSTEM WITH POWER SUPPLY, CABINET, DISK DRIVE CABLE, J&M  
NEW DISK CONTROLLER JPD-CP WITH J-DOS, RS-DOS OPERATING  
SYSTEMS. **\$469.95**

\* Specify What CONTROLLER You Want J&M, or RADIO SHACK

THINLINE DOUBLE SIDED  
DOUBLE DENSITY 40 TRACKS **\$129.95**

**Verbatim Diskettes**

Single Sided Double Density **\$ 24.00**  
Double Sided Double Density **\$ 24.00**

**Controllers**

J&M JPD-CP WITH J-DOS **\$139.95**  
WITH J-DOS, RS-DOS **\$159.95**  
RADIO SHACK 1.1 **\$134.95**

RADIO SHACK Disk CONTROLLER 1.1 **\$134.95**

**Disk Drive Cables**

Cable for One Drive **\$ 19.95**  
Cable for Two Drives **\$ 24.95**

**MISC**

64K UPGRADE **\$ 29.95**  
FOR C,D,E,F, AND COCO 11  
RADIO SHACK BASIC 1.2 **\$ 24.95**  
RADIO SHACK DISK BASIC 1.1 **\$ 24.95**

DISK DRIVE CABINET FOR A  
SINGLE DRIVE **\$ 49.95**  
DISK DRIVE CABINET FOR TWO  
THINLINE DRIVES **\$ 69.95**

**PRINTERS**

EPSON LX-80 **\$289.95**  
EPSON MX-70 **\$125.95**  
EPSON MX-100 **\$495.95**

**ACCESSORIES FOR EPSON**

8148 2K SERIAL BOARD **\$ 89.95**  
8149 32K EXPAND TO 128K **\$169.95**  
EPSON MX-RX-80 RIBBONS **\$ 7.95**  
EPSON LX-80 RIBBONS **\$ 5.95**  
TRACTOR UNITS FOR LX-80 **\$ 39.95**  
CABLES & OTHER INTERFACES  
CALL FOR PRICING

**DATA-COMP**  
5900 Cassandra Smith Rd.  
Hixson, TN 37343



**SHIPPING**  
USA ADD 2%  
FOREIGN ADD 5%  
MIN. \$2.50

**(615)842-4600**  
For Ordering  
Telex 5108008630



# An Ace of a System in Spades! The New MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the \$500.00 OS-9 C compiler! This offer won't last forever!

## NOT 128K, NOT 512K FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SKDOS™. SKDOS is a single user, single tasking system that takes up where "FLEX" left off. SKDOS is actually a 68XXX FLEX type system (Not a TSC product.)

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed whiz on disk I/O. Fact is the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks.

System Includes OS-9 68K or SKDOS - Your Choice Specifications:

CPU	MC68008	12 Mhz
RAM	768K	256K Chips
	No Wait States	
PORTS	4 - RS232	MC86881 DUART
	2 - 8 bit Parallel	MC8821 PIA
CLOCK	MK48T02	Real Time Clock Bat. BU
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU  
Battery B/U - and \$850.00 OS-9 Profes-  
sional with C compiler included!

**\*\$400.00**

See Mustang-02 Ad - page 5  
for trade-in details



MUSTANG-08

LOOK

Seconds 32 bit Register  
Integer Long

Other 68008 8 Mhz OS-9 68K...18.0...9.0

MUSTANG-08 10 Mhz OS-9 68K...9.8...6.3  
MinIn()

C Benchmark Loop

```
/* Init i; */
register long i;
for (i=0; i < 999999; ++i);
```

Now even faster!  
with 12 Mhz CPU

C Compile times: OS-9 68K	Hard Disk
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68008 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec



25 Megabyte  
Hard Disk System

**\$2,398.90**

Complete with PROFESSIONAL OS-9  
includes the \$500.00 C compiler, PC  
style cabinet, heavy duty power supply,  
5" DDDS 80 track floppy, 25 MegByte  
Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabyte of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC

MUSTANG-08 is a trademark of CPI

**Data-Comp Division**



A Decade of Quality Service™  
Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road  
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, TN 37343

\* Those with SW/PC Hixsonity FLEX 5" - Call for special info.